

Oracle® Spatial

User's Guide and Reference

10g Release 2 (10.2)

B14255-03

March 2006

Provides usage and reference information for indexing and storing spatial data and for developing spatial applications using Oracle Spatial and Oracle Locator.

Oracle Spatial User's Guide and Reference, 10g Release 2 (10.2)

B14255-03

Copyright © 1999, 2006, Oracle. All rights reserved.

Primary Author: Chuck Murray

Contributors: Dan Abugov, Nicole Alexander, Bruce Blackwell, Janet Blowney, Dan Geringer, Albert Godfrind, Mike Horhammer, Ravi Kothuri, Richard Pitts, Vishal Rao, Siva Ravada, Jack Wang, Ji Yang

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface	xix
Audience	xix
Documentation Accessibility	xix
Related Documents	xx
Conventions	xxi
What's New in Oracle Spatial?	xxiii
Coordinate System Support Based on EPSG Model	xxiii
New SDO_GEOMETRY Methods and Constructors	xxiv
New min_resolution and max_resolution Keywords	xxiv
New sdo_dml_batch_size Parameter	xxv
New Geocoding Subprograms	xxv
New Utility Subprograms	xxv
U.S. National Grid Support	xxvi
Unknown and NaC Coordinate Reference Systems	xxvi
Spatial Routing Engine	xxvi
LRS_INTERSECTION Function	xxvi
Part I Conceptual and Usage Information	
1 Spatial Concepts	
1.1 What Is Oracle Spatial?	1-2
1.2 Object-Relational Model	1-2
1.3 Introduction to Spatial Data	1-3
1.4 Geometry Types	1-3
1.5 Data Model	1-4
1.5.1 Element	1-4
1.5.2 Geometry	1-5
1.5.3 Layer	1-5
1.5.4 Coordinate System	1-5
1.5.5 Tolerance	1-6
1.5.5.1 Tolerance in the Geometry Metadata for a Layer	1-6
1.5.5.2 Tolerance as an Input Parameter	1-7
1.6 Query Model	1-8
1.7 Indexing of Spatial Data	1-9

1.7.1	R-Tree Indexing.....	1-9
1.7.2	R-Tree Quality	1-10
1.8	Spatial Relationships and Filtering	1-11
1.9	Spatial Operators, Procedures, and Functions	1-13
1.10	Spatial Aggregate Functions	1-14
1.10.1	SDOAGGRTYPE Object Type.....	1-14
1.11	Geocoding	1-15
1.12	Spatial Java Application Programming Interface	1-15
1.13	MDDATA Schema	1-16
1.14	Performance and Tuning Information.....	1-16
1.15	Open Geospatial Consortium (OGC) Conformance.....	1-16
1.16	Spatial Release (Version) Number.....	1-16
1.17	Spatial Application Hardware Requirement Considerations	1-17
1.18	Spatial Error Messages	1-17
1.19	Spatial Examples	1-17
1.20	README File for Spatial and Related Features.....	1-18

2 Spatial Data Types and Metadata

2.1	Simple Example: Inserting, Indexing, and Querying Spatial Data.....	2-1
2.2	SDO_GEOMETRY Object Type	2-5
2.2.1	SDO_GTYPE.....	2-5
2.2.2	SDO_SRID.....	2-7
2.2.3	SDO_POINT	2-7
2.2.4	SDO_ELEM_INFO.....	2-7
2.2.5	SDO_ORDINATES	2-10
2.2.6	Usage Considerations	2-11
2.3	SDO_GEOMETRY Methods.....	2-11
2.4	SDO_GEOMETRY Constructors.....	2-13
2.5	Geometry Examples.....	2-14
2.5.1	Rectangle.....	2-14
2.5.2	Polygon with a Hole.....	2-15
2.5.3	Compound Line String	2-17
2.5.4	Compound Polygon	2-19
2.5.5	Point.....	2-20
2.5.6	Oriented Point.....	2-21
2.5.7	Type 0 (Zero) Element.....	2-23
2.5.8	Several Geometry Types	2-25
2.6	Geometry Metadata Views	2-29
2.6.1	TABLE_NAME.....	2-30
2.6.2	COLUMN_NAME.....	2-30
2.6.3	DIMINFO	2-30
2.6.4	SRID	2-31
2.7	Spatial Index-Related Structures	2-31
2.7.1	Spatial Index Views	2-31
2.7.1.1	xxx_SDO_INDEX_INFO Views.....	2-31
2.7.1.2	xxx_SDO_INDEX_METADATA Views	2-32
2.7.2	Spatial Index Table Definition	2-34

2.7.3	R-Tree Index Sequence Object	2-35
2.8	Unit of Measurement Support	2-35

3 Loading Spatial Data

3.1	Bulk Loading	3-1
3.1.1	Bulk Loading SDO_GEOMETRY Objects	3-1
3.1.2	Bulk Loading Point-Only Data in SDO_GEOMETRY Objects.....	3-3
3.2	Transactional Insert Operations Using SQL	3-3

4 Indexing and Querying Spatial Data

4.1	Creating a Spatial Index.....	4-1
4.1.1	Indexing Geodetic Data	4-2
4.1.2	Constraining Data to a Geometry Type.....	4-2
4.1.3	Creating a Cross-Schema Index.....	4-3
4.1.4	Using Partitioned Spatial Indexes	4-3
4.1.5	Exchanging Partitions Including Indexes	4-5
4.1.6	Export and Import Considerations with Spatial Indexes and Data	4-5
4.1.7	Distributed Transactions and Spatial Index Consistency	4-6
4.2	Querying Spatial Data	4-6
4.2.1	Spatial Query	4-7
4.2.1.1	Primary Filter Operator	4-8
4.2.1.2	Primary and Secondary Filter Operator.....	4-9
4.2.1.3	Within-Distance Operator	4-9
4.2.1.4	Nearest Neighbor Operator	4-11
4.2.1.5	Spatial Functions.....	4-11
4.2.2	Spatial Join	4-11
4.2.3	Cross-Schema Operator Invocation	4-12

5 Geocoding Address Data

5.1	Concepts for Geocoding.....	5-1
5.1.1	Address Representation.....	5-1
5.1.2	Match Modes	5-2
5.1.3	Match Codes	5-3
5.1.4	Error Messages for Output Geocoded Addresses	5-4
5.2	Data Types for Geocoding.....	5-4
5.2.1	SDO_GEO_ADDR Type	5-4
5.2.2	SDO_ADDR_ARRAY Type.....	5-7
5.2.3	SDO_KEYWORDARRAY Type.....	5-7
5.3	Using the Geocoding Capabilities	5-7
5.4	Geocoding from a Place Name.....	5-8
5.5	Data Structures for Geocoding.....	5-9
5.5.1	GC_AREA_<suffix> Table	5-10
5.5.2	GC_COUNTRY_PROFILE Table.....	5-11
5.5.3	GC_INTERSECTION_<suffix> Table	5-13
5.5.4	GC_POI_<suffix> Table.....	5-14
5.5.5	GC_POSTAL_CODE_<suffix> Table.....	5-15

5.5.6	GC_ROAD_<suffix> Table.....	5-16
5.5.7	GC_ROAD_SEGMENT_<suffix> Table	5-18

6 Coordinate Systems (Spatial Reference Systems)

6.1	Terms and Concepts	6-1
6.1.1	Coordinate System (Spatial Reference System)	6-1
6.1.2	Cartesian Coordinates.....	6-2
6.1.3	Geodetic Coordinates (Geographic Coordinates).....	6-2
6.1.4	Projected Coordinates	6-2
6.1.5	Local Coordinates	6-2
6.1.6	Geodetic Datum	6-2
6.1.7	Transformation.....	6-2
6.2	Geodetic Coordinate Support	6-2
6.2.1	Geodesy and Two-Dimensional Geometry	6-3
6.2.2	Choosing a Geodetic or Projected Coordinate System.....	6-3
6.2.3	Geodetic MBRs.....	6-3
6.2.4	Other Considerations and Requirements with Geodetic Data	6-5
6.3	Local Coordinate Support.....	6-6
6.4	EPSG Model and Spatial	6-6
6.5	TFM_PLAN Object Type	6-7
6.6	Coordinate Systems Data Structures.....	6-8
6.6.1	SDO_COORD_AXES Table	6-8
6.6.2	SDO_COORD_AXIS_NAMES Table	6-9
6.6.3	SDO_COORD_OP_METHODS Table	6-9
6.6.4	SDO_COORD_OP_PARAM_USE Table.....	6-10
6.6.5	SDO_COORD_OP_PARAM_VALS Table	6-10
6.6.6	SDO_COORD_OP_PARAMS Table.....	6-11
6.6.7	SDO_COORD_OP_PATHS Table	6-11
6.6.8	SDO_COORD_OPS Table.....	6-12
6.6.9	SDO_COORD_REF_SYS Table	6-13
6.6.10	SDO_COORD_REF_SYSTEM View	6-15
6.6.11	SDO_COORD_SYS Table	6-15
6.6.12	SDO_CRS_COMPOUND View	6-15
6.6.13	SDO_CRS_ENGINEERING View	6-16
6.6.14	SDO_CRS_GEOCENTRIC View	6-16
6.6.15	SDO_CRS_GEOGRAPHIC2D View.....	6-17
6.6.16	SDO_CRS_GEOGRAPHIC3D View.....	6-17
6.6.17	SDO_CRS_PROJECTED View	6-18
6.6.18	SDO_CRS_VERTICAL View	6-18
6.6.19	SDO_DATUM_ENGINEERING View	6-19
6.6.20	SDO_DATUM_GEODETTIC View	6-20
6.6.21	SDO_DATUM_VERTICAL View	6-20
6.6.22	SDO_DATUMS Table.....	6-21
6.6.23	SDO_ELLIPSOIDS Table	6-22
6.6.24	SDO_PREFERRED_OPS_SYSTEM Table.....	6-23
6.6.25	SDO_PREFERRED_OPS_USER Table	6-23
6.6.26	SDO_PRIME_MERIDIANS Table	6-24

6.6.27	SDO_UNITS_OF_MEASURE Table.....	6-24
6.7	Legacy Tables and Views.....	6-25
6.7.1	MDSYS.CS_SRS Table.....	6-26
6.7.1.1	Well-Known Text (WKT).....	6-26
6.7.1.2	Procedures for Updating the Well-Known Text	6-28
6.7.2	MDSYS.SDO_ANGLE_UNITS View	6-29
6.7.3	MDSYS.SDO_AREA_UNITS View	6-29
6.7.4	MDSYS.SDO_DATUMS_OLD_FORMAT and SDO_DATUMS_OLD_SNAPSHOT Tables 6-30	
6.7.5	MDSYS.SDO_DIST_UNITS View.....	6-32
6.7.6	MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT and SDO_ELLIPSOIDS_OLD_ SNAPSHOT Tables 6-32	
6.7.7	MDSYS.SDO_PROJECTIONS_OLD_FORMAT and SDO_PROJECTIONS_OLD_ SNAPSHOT Tables 6-33	
6.8	Creating a User-Defined Coordinate Reference System	6-34
6.8.1	Creating a Geodetic CRS.....	6-35
6.8.2	Creating a Projected CRS.....	6-36
6.9	Notes and Restrictions with Coordinate Systems Support.....	6-40
6.9.1	Different Coordinate Systems for Geometries with Operators and Functions	6-40
6.9.2	3D LRS Functions Not Supported with Geodetic Data.....	6-40
6.9.3	Functions Supported by Approximations with Geodetic Data	6-40
6.9.4	Unknown CRS and NaC Coordinate Reference Systems	6-40
6.10	U.S. National Grid Support	6-41
6.11	Example of Coordinate System Transformation	6-41

7 Linear Referencing System

7.1	Terms and Concepts	7-1
7.1.1	Geometric Segments (LRS Segments)	7-2
7.1.2	Shape Points	7-2
7.1.3	Direction of a Geometric Segment	7-3
7.1.4	Measure (Linear Measure).....	7-3
7.1.5	Offset.....	7-3
7.1.6	Measure Populating	7-3
7.1.7	Measure Range of a Geometric Segment.....	7-5
7.1.8	Projection	7-5
7.1.9	LRS Point.....	7-5
7.1.10	Linear Features.....	7-5
7.1.11	Measures with Multiline Strings and Polygons with Holes.....	7-5
7.2	LRS Data Model	7-6
7.3	Indexing of LRS Data.....	7-7
7.4	3D Formats of LRS Functions.....	7-7
7.5	LRS Operations.....	7-8
7.5.1	Defining a Geometric Segment	7-8
7.5.2	Redefining a Geometric Segment	7-9
7.5.3	Clipping a Geometric Segment.....	7-9
7.5.4	Splitting a Geometric Segment	7-10
7.5.5	Concatenating Geometric Segments	7-10

7.5.6	Scaling a Geometric Segment	7-11
7.5.7	Offsetting a Geometric Segment.....	7-12
7.5.8	Locating a Point on a Geometric Segment	7-12
7.5.9	Projecting a Point onto a Geometric Segment	7-13
7.5.10	Converting LRS Geometries.....	7-14
7.6	Tolerance Values with LRS Functions	7-15
7.7	Example of LRS Functions.....	7-15

8 Spatial Analysis and Mining

8.1	Spatial Information and Data Mining Applications	8-1
8.2	Spatial Binning for Detection of Regional Patterns.....	8-3
8.3	Materializing Spatial Correlation	8-3
8.4	Colocation Mining	8-4
8.5	Spatial Clustering.....	8-4
8.6	Location Prospecting	8-5

9 Extending Spatial Indexing Capabilities

9.1	SDO_GEOMETRY Objects in User-Defined Type Definitions	9-1
9.2	SDO_GEOMETRY Objects in Function-Based Indexes.....	9-3
9.2.1	Example: Function with Standard Types	9-3
9.2.2	Example: Function with a User-Defined Object Type.....	9-4

Part II Reference Information

10 SQL Statements for Indexing Spatial Data

ALTER INDEX	10-2
ALTER INDEX REBUILD.....	10-4
ALTER INDEX RENAME TO	10-7
CREATE INDEX.....	10-8
DROP INDEX	10-12

11 Spatial Operators

SDO_ANYINTERACT	11-3
SDO_CONTAINS	11-5
SDO_COVEREDBY	11-6
SDO_COVERS	11-7
SDO_EQUAL.....	11-8
SDO_FILTER.....	11-9
SDO_INSIDE	11-12
SDO_JOIN	11-13
SDO_NN	11-16
SDO_NN_DISTANCE.....	11-20
SDO_ON.....	11-22

SDO_OVERLAPBDYDISJOINT.....	11-23
SDO_OVERLAPBDYINTERSECT.....	11-25
SDO_OVERLAPS.....	11-27
SDO_RELATE.....	11-29
SDO_TOUCH.....	11-33
SDO_WITHIN_DISTANCE.....	11-35

12 Spatial Aggregate Functions

SDO_AGGR_CENTROID.....	12-2
SDO_AGGR_CONCAT_LINES.....	12-3
SDO_AGGR_CONVEXHULL.....	12-5
SDO_AGGR_LRS_CONCAT.....	12-6
SDO_AGGR_MBR.....	12-8
SDO_AGGR_UNION.....	12-9

13 SDO_CS Package (Coordinate System Transformation)

SDO_CS.ADD_PREFERENCE_FOR_OP.....	13-4
SDO_CS.CONVERT_NADCON_TO_XML.....	13-6
SDO_CS.CONVERT_NTV2_TO_XML.....	13-8
SDO_CS.CONVERT_XML_TO_NADCON.....	13-10
SDO_CS.CONVERT_XML_TO_NTV2.....	13-12
SDO_CS.CREATE_CONCATENATED_OP.....	13-14
SDO_CS.CREATE_OBVIOUS_EPSG_RULES.....	13-15
SDO_CS.CREATE_PREF_CONCATENATED_OP.....	13-16
SDO_CS.DELETE_ALL_EPSG_RULES.....	13-18
SDO_CS.DELETE_OP.....	13-19
SDO_CS.DETERMINE_CHAIN.....	13-20
SDO_CS.DETERMINE_DEFAULT_CHAIN.....	13-22
SDO_CS.FIND_GEOG_CRIS.....	13-23
SDO_CS.FIND_PROJ_CRIS.....	13-25
SDO_CS.FROM_OGC_SIMPLEFEATURE_SRS.....	13-27
SDO_CS.FROM_USNG.....	13-28
SDO_CS.MAP_EPSG_SRID_TO_ORACLE.....	13-29
SDO_CS.MAP_ORACLE_SRID_TO_EPSG.....	13-30
SDO_CS.REVOKE_PREFERENCE_FOR_OP.....	13-31
SDO_CS.TO_OGC_SIMPLEFEATURE_SRS.....	13-32
SDO_CS.TO_USNG.....	13-33
SDO_CS.TRANSFORM.....	13-35
SDO_CS.TRANSFORM_LAYER.....	13-38
SDO_CS.UPDATE_WKTS_FOR_ALL_EPSG_CRIS.....	13-40
SDO_CS.UPDATE_WKTS_FOR_EPSG_CRIS.....	13-41

SDO_CS.UPDATE_WKTS_FOR_EPSG_DATUM.....	13-42
SDO_CS.UPDATE_WKTS_FOR_EPSG_ELLIPS.....	13-43
SDO_CS.UPDATE_WKTS_FOR_EPSG_OP	13-44
SDO_CS.UPDATE_WKTS_FOR_EPSG_PARAM.....	13-45
SDO_CS.UPDATE_WKTS_FOR_EPSG_PM.....	13-46
SDO_CS.VALIDATE_WKT	13-47
SDO_CS.VIEWPORT_TRANSFORM	13-48

14 SDO_GCDR Package (Geocoding)

SDO_GCDR.GEOCODE	14-2
SDO_GCDR.GEOCODE_ADDR	14-3
SDO_GCDR.GEOCODE_ADDR_ALL	14-5
SDO_GCDR.GEOCODE_ALL	14-7
SDO_GCDR.GEOCODE_AS_GEOMETRY	14-9
SDO_GCDR.REVERSE_GEOCODE.....	14-10

15 SDO_GEOM Package (Geometry)

SDO_GEOM.RELATE	15-3
SDO_GEOM.SDO_ARC_DENSIFY.....	15-6
SDO_GEOM.SDO_AREA.....	15-8
SDO_GEOM.SDO_BUFFER	15-10
SDO_GEOM.SDO_CENTROID	15-13
SDO_GEOM.SDO_CONVEXHULL.....	15-15
SDO_GEOM.SDO_DIFFERENCE	15-17
SDO_GEOM.SDO_DISTANCE.....	15-19
SDO_GEOM.SDO_INTERSECTION	15-21
SDO_GEOM.SDO_LENGTH	15-23
SDO_GEOM.SDO_MAX_MBR_ORDINATE	15-25
SDO_GEOM.SDO_MBR	15-27
SDO_GEOM.SDO_MIN_MBR_ORDINATE	15-29
SDO_GEOM.SDO_POINTONSURFACE.....	15-31
SDO_GEOM.SDO_UNION	15-33
SDO_GEOM.SDO_XOR.....	15-35
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT.....	15-37
SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT	15-41
SDO_GEOM.WITHIN_DISTANCE	15-43

16 SDO_LRS Package (Linear Referencing System)

SDO_LRS.CLIP_GEOM_SEGMENT	16-5
SDO_LRS.CONCATENATE_GEOM_SEGMENTS	16-7
SDO_LRS.CONNECTED_GEOM_SEGMENTS.....	16-10
SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY.....	16-12

SDO_LRS.CONVERT_TO_LRS_GEOM.....	16-14
SDO_LRS.CONVERT_TO_LRS_LAYER.....	16-16
SDO_LRS.CONVERT_TO_STD_DIM_ARRAY	16-18
SDO_LRS.CONVERT_TO_STD_GEOM	16-19
SDO_LRS.CONVERT_TO_STD_LAYER.....	16-20
SDO_LRS.DEFINE_GEOM_SEGMENT	16-22
SDO_LRS.DYNAMIC_SEGMENT	16-25
SDO_LRS.FIND_LRS_DIM_POS.....	16-27
SDO_LRS.FIND_MEASURE	16-28
SDO_LRS.FIND_OFFSET	16-30
SDO_LRS.GEOM_SEGMENT_END_MEASURE	16-32
SDO_LRS.GEOM_SEGMENT_END_PT	16-33
SDO_LRS.GEOM_SEGMENT_LENGTH.....	16-34
SDO_LRS.GEOM_SEGMENT_START_MEASURE	16-35
SDO_LRS.GEOM_SEGMENT_START_PT	16-36
SDO_LRS.GET_MEASURE	16-37
SDO_LRS.GET_NEXT_SHAPE_PT.....	16-38
SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE	16-40
SDO_LRS.GET_PREV_SHAPE_PT	16-42
SDO_LRS.GET_PREV_SHAPE_PT_MEASURE.....	16-44
SDO_LRS.IS_GEOM_SEGMENT_DEFINED	16-46
SDO_LRS.IS_MEASURE_DECREASING	16-47
SDO_LRS.IS_MEASURE_INCREASING	16-48
SDO_LRS.IS_SHAPE_PT_MEASURE.....	16-49
SDO_LRS.LOCATE_PT.....	16-51
SDO_LRS.LRS_INTERSECTION.....	16-53
SDO_LRS.MEASURE_RANGE.....	16-55
SDO_LRS.MEASURE_TO_PERCENTAGE	16-56
SDO_LRS.OFFSET_GEOM_SEGMENT	16-58
SDO_LRS.PERCENTAGE_TO_MEASURE	16-61
SDO_LRS.PROJECT_PT.....	16-63
SDO_LRS.REDEFINE_GEOM_SEGMENT.....	16-65
SDO_LRS.RESET_MEASURE	16-67
SDO_LRS.REVERSE_GEOMETRY.....	16-69
SDO_LRS.REVERSE_MEASURE.....	16-71
SDO_LRS.SET_PT_MEASURE	16-73
SDO_LRS.SPLIT_GEOM_SEGMENT	16-76
SDO_LRS.TRANSLATE_MEASURE	16-78
SDO_LRS.VALID_GEOM_SEGMENT	16-80
SDO_LRS.VALID_LRS_PT	16-81
SDO_LRS.VALID_MEASURE	16-82

	SDO_LRS.VALIDATE_LRS_GEOMETRY	16-84
17	SDO_MIGRATE Package (Upgrading)	
	SDO_MIGRATE.TO_CURRENT	17-2
18	SDO_SAM Package (Spatial Analysis and Mining)	
	SDO_SAM.AGGREGATES_FOR_GEOMETRY	18-2
	SDO_SAM.AGGREGATES_FOR_LAYER	18-4
	SDO_SAM.BIN_GEOMETRY	18-6
	SDO_SAM.BIN_LAYER.....	18-8
	SDO_SAM.COLOCATED_REFERENCE_FEATURES.....	18-10
	SDO_SAM.SIMPLIFY_GEOMETRY	18-12
	SDO_SAM.SIMPLIFY_LAYER	18-14
	SDO_SAM.SPATIAL_CLUSTERS.....	18-16
	SDO_SAM.TILED_AGGREGATES.....	18-17
	SDO_SAM.TILED_BINS	18-20
19	SDO_TUNE Package (Tuning)	
	SDO_TUNE.AVERAGE_MBR	19-2
	SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE.....	19-4
	SDO_TUNE.EXTENT_OF.....	19-7
	SDO_TUNE.MIX_INFO	19-8
	SDO_TUNE.QUALITY_DEGRADATION.....	19-10
20	SDO_UTIL Package (Utility)	
	SDO_UTIL.APPEND.....	20-3
	SDO_UTIL.CIRCLE_POLYGON	20-4
	SDO_UTIL.CONCAT_LINES	20-6
	SDO_UTIL.CONVERT_UNIT.....	20-8
	SDO_UTIL.ELLIPSE_POLYGON	20-9
	SDO_UTIL.EXTRACT	20-11
	SDO_UTIL.FROM_WKBGEOMETRY	20-13
	SDO_UTIL.FROM_WKTGEOMETRY	20-15
	SDO_UTIL.GETNUMELEM	20-17
	SDO_UTIL.GETNUMVERTICES	20-18
	SDO_UTIL.GETVERTICES.....	20-19
	SDO_UTIL.INITIALIZE_INDEXES_FOR_TTS	20-21
	SDO_UTIL.POINT_AT_BEARING	20-22
	SDO_UTIL.POLYGONTOLINE	20-24
	SDO_UTIL.PREPARE_FOR_TTS.....	20-25
	SDO_UTIL.RECTIFY_GEOMETRY	20-26
	SDO_UTIL.REMOVE_DUPLICATE_VERTICES	20-27

SDO_UTIL.REVERSE_LINestring.....	20-29
SDO_UTIL.SIMPLIFY	20-30
SDO_UTIL.TO_GMLGEOMETRY	20-33
SDO_UTIL.TO_WKBGEOMETRY	20-39
SDO_UTIL.TO_WKTGEOMETRY	20-41
SDO_UTIL.VALIDATE_WKBGEOMETRY	20-43
SDO_UTIL.VALIDATE_WKTGEOMETRY	20-45

Part III Supplementary Information

A Installation, Compatibility, and Upgrade

A.1	Upgrading LRS Data	A-1
A.2	Ensuring That GeoRaster Works Properly After an Upgrade	A-1

B Oracle Locator

C Routing Engine

C.1	Deploying and Configuring the Routing Engine	C-2
C.2	Routing Engine XML API	C-3
C.2.1	Route Request and Response Examples	C-5
C.2.2	Route Request DTD	C-9
C.2.2.1	route_request Element	C-10
C.2.2.2	route_request Attributes	C-10
C.2.2.3	input_location Element	C-12
C.2.2.4	pre_geocoded_location Element	C-12
C.2.2.5	longitude_latitude_location Element	C-12
C.2.3	Route Response DTD	C-12
C.2.4	Batch Route Request and Response Examples	C-13
C.2.5	Batch Route Request DTD	C-16
C.2.5.1	batch_route_request Element	C-16
C.2.5.2	batch_route_request Attributes	C-17
C.2.6	Batch Route Response DTD	C-17
C.2.7	Geocoding Request and Response DTDs	C-18
C.2.7.1	Geocoding Request DTD	C-18
C.2.7.2	Geocoding Response DTD	C-19
C.3	Data Structures Used by the Routing Engine	C-20
C.3.1	EDGE Table	C-20
C.3.2	NODE Table	C-21
C.3.3	PARTITION Table	C-21
C.3.4	SIGN_POST Table	C-21

D Complex Spatial Queries: Examples

D.1	Tables Used in the Examples	D-1
D.2	SDO_WITHIN_DISTANCE Examples	D-2
D.3	SDO_NN Examples	D-3

D.4	SDO_AGGR_UNION Example	D-5
-----	------------------------------	-----

Glossary

Index

List of Examples

2-1	Simple Example: Inserting, Indexing, and Querying Spatial Data.....	2-2
2-2	SDO_GEOMETRY Methods.....	2-12
2-3	SDO_GEOMETRY Constructors to Create Geometries	2-13
2-4	SQL Statement to Insert a Rectangle.....	2-15
2-5	SQL Statement to Insert a Polygon with a Hole.....	2-16
2-6	SQL Statement to Insert a Compound Line String.....	2-18
2-7	SQL Statement to Insert a Compound Polygon	2-20
2-8	SQL Statement to Insert a Point-Only Geometry	2-21
2-9	Query for Point-Only Geometry Based on a Coordinate Value	2-21
2-10	SQL Statement to Insert an Oriented Point Geometry	2-22
2-11	SQL Statement to Insert an Oriented Multipoint Geometry	2-23
2-12	SQL Statement to Insert a Geometry with a Type 0 Element.....	2-25
2-13	SQL Statements to Insert Various Geometries.....	2-25
3-1	Control File for a Bulk Load of Cola Market Geometries	3-1
3-2	Control File for a Bulk Load of Polygons.....	3-2
3-3	Control File for a Bulk Load of Point-Only Data.....	3-3
3-4	Procedure to Perform a Transactional Insert Operation	3-4
3-5	PL/SQL Block Invoking a Procedure to Insert a Geometry	3-4
4-1	Primary Filter with a Temporary Query Window	4-8
4-2	Primary Filter with a Transient Instance of the Query Window	4-8
4-3	Primary Filter with a Stored Query Window	4-8
4-4	Secondary Filter Using a Temporary Query Window	4-9
4-5	Secondary Filter Using a Stored Query Window	4-9
5-1	Geocoding, Returning Address Object and Specific Attributes.....	5-6
5-2	Geocoding from a Place Name and Country	5-8
5-3	Geocoding from a Place Name, Country, and Other Fields	5-8
6-1	Using a Geodetic MBR	6-4
6-2	Creating a User-Defined Geodetic Coordinate Reference System	6-35
6-3	Inserting a Row into the SDO_COORD_SYS Table.....	6-36
6-4	Creating a User-Defined Projected Coordinate Reference System.....	6-37
6-5	Inserting a Row into the SDO_COORD_OPS Table	6-38
6-6	Inserting a Row into the SDO_COORD_OP_PARAM_VALS Table.....	6-38
6-7	Simplified Example of Coordinate System Transformation.....	6-42
6-8	Output of SELECT Statements in Coordinate System Transformation Example	6-44
7-1	Including LRS Measure Dimension in Spatial Metadata	7-6
7-2	Simplified Example: Highway	7-17
7-3	Simplified Example: Output of SELECT Statements.....	7-20
C-1	Route Request with Specified Addresses.....	C-5
C-2	Route Response with Specified Addresses	C-5
C-3	Route Request with Specified Longitude/Latitude Points.....	C-7
C-4	Route Response with Specified Longitude/Latitude Points	C-7
C-5	Route Request with Previously Geocoded Locations.....	C-8
C-6	Route Response with Previously Geocoded Locations	C-9
C-7	Batch Route Request with Specified Addresses	C-13
C-8	Batch Route Response with Specified Addresses	C-14
C-9	Batch Route Request with Previously Geocoded Locations	C-14
C-10	Batch Route Response with Previously Geocoded Locations	C-15
D-1	Finding All Cities Within a Distance of a Highway	D-2
D-2	Finding All Highways Within a Distance of a City	D-2
D-3	Finding the Cities Nearest to a Highway	D-3
D-4	Finding the Cities Above a Specified Population Nearest to a Highway.....	D-4
D-5	Performing Aggregate Union of All Counties in Texas	D-5

List of Figures

1-1	Geometric Types	1-4
1-2	Query Model.....	1-8
1-3	MBR Enclosing a Geometry.....	1-9
1-4	R-Tree Hierarchical Index on MBRs.....	1-10
1-5	The Nine-Intersection Model	1-11
1-6	Topological Relationships.....	1-12
1-7	Distance Buffers for Points, Lines, and Polygons.....	1-13
1-8	Tolerance in an Aggregate Union Operation.....	1-15
2-1	Areas of Interest for the Simple Example.....	2-2
2-2	Rectangle	2-15
2-3	Polygon with a Hole	2-16
2-4	Compound Line String.....	2-18
2-5	Compound Polygon	2-19
2-6	Point-Only Geometry	2-20
2-7	Oriented Point Geometry.....	2-22
2-8	Geometry with Type 0 (Zero) Element	2-24
4-1	Geometries with MBRs	4-7
4-2	Layer with a Query Window	4-7
7-1	Geometric Segment.....	7-2
7-2	Describing a Point Along a Segment with a Measure and an Offset	7-3
7-3	Measures, Distances, and Their Mapping Relationship.....	7-4
7-4	Measure Populating of a Geometric Segment	7-4
7-5	Measure Populating with Disproportional Assigned Measures	7-4
7-6	Linear Feature, Geometric Segments, and LRS Points	7-5
7-7	Creating a Geometric Segment	7-6
7-8	Defining a Geometric Segment	7-8
7-9	Redefining a Geometric Segment	7-9
7-10	Clipping, Splitting, and Concatenating Geometric Segments.....	7-9
7-11	Measure Assignment in Geometric Segment Operations.....	7-10
7-12	Segment Direction with Concatenation.....	7-11
7-13	Scaling a Geometric Segment.....	7-11
7-14	Offsetting a Geometric Segment.....	7-12
7-15	Locating a Point Along a Segment with a Measure and an Offset	7-12
7-16	Ambiguity in Location Referencing with Offsets.....	7-13
7-17	Multiple Projection Points	7-13
7-18	Conversion from Standard to LRS Line String.....	7-14
7-19	Segment for Clip Operation Affected by Tolerance.....	7-15
7-20	Simplified LRS Example: Highway	7-16
8-1	Spatial Mining and Oracle Data Mining.....	8-2
15-1	Arc Tolerance.....	15-7
15-2	SDO_GEOM.SDO_DIFFERENCE	15-18
15-3	SDO_GEOM.SDO_INTERSECTION	15-22
15-4	SDO_GEOM.SDO_UNION	15-34
15-5	SDO_GEOM.SDO_XOR.....	15-36
16-1	Translating a Geometric Segment	16-78
20-1	Simplification of a Geometry	20-32
C-1	Basic Flow of Action with the Spatial Routing Engine.....	C-1

List of Tables

2-1	Valid SDO_GTYPE Values	2-6
2-2	Values and Semantics in SDO_ELEM_INFO	2-9
2-3	SDO_GEOMETRY Methods.....	2-11
2-4	Columns in the xxx_SDO_INDEX_INFO Views.....	2-32
2-5	Columns in the xxx_SDO_INDEX_METADATA Views	2-32
2-6	Columns in an R-Tree Spatial Index Data Table	2-34
5-1	Attributes for Formal Address Representation	5-1
5-2	Match Modes for Geocoding Operations	5-2
5-3	Match Codes for Geocoding Operations	5-3
5-4	Geocoded Address Error Message Interpretation	5-4
5-5	SDO_GEO_ADDR Type Attributes.....	5-5
5-6	GC_AREA_<suffix> Table.....	5-10
5-7	GC_COUNTRY_PROFILE Table.....	5-11
5-8	GC_INTERSECTION_<suffix> Table	5-13
5-9	GC_POI_<suffix> Table	5-14
5-10	GC_POSTAL_CODE_<suffix> Table.....	5-15
5-11	GC_ROAD_<suffix> Table	5-16
5-12	GC_ROAD_SEGMENT_<suffix> Table	5-18
6-1	SDO_COORD_AXES Table	6-9
6-2	SDO_COORD_AXIS_NAMES Table	6-9
6-3	SDO_COORD_OP_METHODS Table.....	6-9
6-4	SDO_COORD_OP_PARAM_USE Table	6-10
6-5	SDO_COORD_OP_PARAM_VALS Table	6-10
6-6	SDO_COORD_OP_PARAMS Table.....	6-11
6-7	SDO_COORD_OP_PATHS Table.....	6-11
6-8	SDO_COORD_OPS Table.....	6-12
6-9	SDO_COORD_REF_SYS Table	6-13
6-10	SDO_COORD_SYS Table.....	6-15
6-11	SDO_CRCS_COMPOUND View	6-15
6-12	SDO_CRCS_ENGINEERING View	6-16
6-13	SDO_CRCS_GEOCENTRIC View.....	6-16
6-14	SDO_CRCS_GEOGRAPHIC2D View.....	6-17
6-15	SDO_CRCS_GEOGRAPHIC3D View.....	6-17
6-16	SDO_CRCS_PROJECTED View	6-18
6-17	SDO_CRCS_VERTICAL View.....	6-19
6-18	SDO_DATUM_ENGINEERING View	6-19
6-19	SDO_DATUM_GEODETIC View	6-20
6-20	SDO_DATUM_VERTICAL View	6-20
6-21	SDO_DATUMS Table.....	6-21
6-22	SDO_ELLIPSOIDS Table	6-22
6-23	SDO_PREFERRED_OPS_SYSTEM Table	6-23
6-24	SDO_PREFERRED_OPS_USER Table	6-24
6-25	SDO_PRIME_MERIDIANS Table	6-24
6-26	SDO_UNITS_OF_MEASURE Table.....	6-24
6-27	MDSYS.CS_SRS Table	6-26
6-28	MDSYS.SDO_ANGLE_UNITS View	6-29
6-29	SDO_AREA_UNITS View	6-30
6-30	MDSYS.SDO_DATUMS_OLD_FORMAT and SDO_DATUMS_OLD_SNAPSHOT Tables..	6-30
6-31	MDSYS.SDO_DIST_UNITS View.....	6-32
6-32	MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT and SDO_ELLIPSOIDS_OLD_SNAPSHOT Tables	6-33
6-33	MDSYS.SDO_PROJECTIONS_OLD_FORMAT and SDO_PROJECTIONS_OLD_	

	SNAPSHOT Tables	6-34
7-1	Highway Features and LRS Counterparts	7-16
10-1	Spatial Index Creation and Usage Statements.....	10-1
11-1	Main Spatial Operators	11-1
11-2	Convenience Operators for SDO_RELATE Operations	11-1
11-3	params Keywords for the SDO_JOIN Operator.....	11-14
11-4	Keywords for the SDO_NN Param Parameter.....	11-16
12-1	Spatial Aggregate Functions	12-1
13-1	Subprograms for Coordinate System Transformation	13-1
13-2	Table to Hold Transformed Layer	13-39
14-1	Subprograms for Geocoding Address Data	14-1
15-1	Geometry Subprograms.....	15-1
16-1	Subprograms for Creating and Editing Geometric Segments.....	16-1
16-2	Subprograms for Querying and Validating Geometric Segments.....	16-2
16-3	Subprograms for Converting Geometric Segments.....	16-3
18-1	Subprograms for Spatial Analysis and Mining	18-1
19-1	Tuning Subprograms.....	19-1
20-1	Spatial Utility Subprograms	20-1
B-1	Spatial Features Supported for Locator	B-2
B-2	Spatial Features Not Supported for Locator	B-2
B-3	Feature Availability with Standard and Enterprise Editions	B-3
C-1	EDGE Table.....	C-20
C-2	NODE Table.....	C-21
C-3	PARTITION Table	C-21
C-4	SIGN_POST Table.....	C-22

Preface

Oracle Spatial User's Guide and Reference provides usage and reference information for indexing and storing spatial data and for developing spatial applications using Oracle Spatial and Oracle Locator.

Oracle Spatial requires the Enterprise Edition of Oracle Database 10g. It is a foundation for the deployment of enterprise-wide spatial information systems, and Web-based and wireless location-based applications requiring complex spatial data management. Oracle Locator is a feature of the Standard and Enterprise Editions of Oracle Database 10g. It offers a subset of Oracle Spatial capabilities (see [Appendix B](#) for a list of Locator features) typically required to support Internet and wireless service applications and partner-based geographic information system (GIS) solutions.

The Standard and Enterprise Editions of Oracle Database 10g have the same basic features. However, several advanced features, such as extended data types, are available only with the Enterprise Edition, and some of these features are optional. For example, to use Oracle Database 10g table partitioning, you must have the Enterprise Edition and the Partitioning Option.

For information about the differences between Oracle Database 10g Standard Edition and Oracle Database 10g Enterprise Edition and the features and options that are available to you, see *Oracle Database New Features*.

Note: The relational geometry model of Oracle Spatial is no longer supported, effective with Oracle release 9.2. Only the object-relational model is supported.

Audience

This guide is intended for anyone who needs to store spatial data in an Oracle database.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be

accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see the following documents:

- *Oracle Spatial GeoRaster*
- *Oracle Spatial Topology and Network Data Models*
- *Oracle Database SQL Reference*
- *Oracle Database Administrator's Guide*
- *Oracle Database Application Developer's Guide - Fundamentals*
- *Oracle Database Error Messages - Spatial messages are in the range of 13000 to 13499.*
- *Oracle Database Performance Tuning Guide*
- *Oracle Database Utilities*
- *Oracle Database Advanced Replication*
- *Oracle Data Cartridge Developer's Guide*

Oracle error message documentation is only available in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

Printed documentation is available for sale in the Oracle Store at

<http://oraclestore.oracle.com/>

To download free release notes, installation documentation, white papers, or other collateral, go to the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

<http://www.oracle.com/technology/membership>

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

<http://www.oracle.com/technology/documentation>

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Oracle Spatial?

This section describes major new and changed Oracle Spatial features for the current release.

Coordinate System Support Based on EPSG Model

Coordinate system support in Spatial is now based on, but is not identical to, the data model and data provided by the European Petroleum Survey Group (EPSG), as explained in [Section 6.4](#). Applications that depend on the data model from previous releases (the "legacy" model) continue to be fully supported; however, the EPSG model bases Spatial on a widely accepted standard and enables greater flexibility for users who want to perform customized datum transformations.

[Chapter 6](#) describes coordinate system concepts, [Section 6.5](#) explains the new TFM_PLAN object type used for transformation plans, and [Section 6.6](#) describes the new EPSG-based tables and views.

The SDO_CS package (described in [Chapter 13](#)) contains the following new subprograms to support the EPSG model:

Subprogram	Description
SDO_CS.ADD_PREFERENCE_FOR_OP	Adds a preference for an operation between a source coordinate system and a target coordinate system.
SDO_CS.CONVERT_NADCON_TO_XML	Converts a NADCON (North American Datum Conversion) grid in ASCII format to an Oracle Spatial XML representation.
SDO_CS.CONVERT_NTV2_TO_XML	Converts an NTV2 (National Transformation Version 2) grid in ASCII format to an Oracle Spatial XML representation.
SDO_CS.CONVERT_XML_TO_NADCON	Converts an Oracle Spatial XML representation of a NADCON (North American Datum Conversion) grid to NADCON ASCII format.
SDO_CS.CONVERT_XML_TO_NTV2	Converts an Oracle Spatial XML representation of an NTV2 (National Transformation Version 2) grid to NTV2 ASCII format.
SDO_CS.CREATE_CONCATENATED_OP	Creates a concatenated operation.
SDO_CS.CREATE_OBVIOUS_EPSG_RULES	Creates a basic set of EPSG rules to be applied in certain transformations.
SDO_CS.CREATE_PREF_CONCATENATED_OP	Creates a concatenated operation, associating it with a transformation plan and making it preferred either systemwide or for a specified use case.

Subprogram	Description
SDO_CS.DELETE_ALL_EPSG_RULES	Deletes the basic set of EPSG rules to be applied in certain transformations.
SDO_CS.DELETE_OP	Deletes a concatenated operation.
SDO_CS.DETERMINE_CHAIN	Returns the query chain, based on the system rule set, to be used in transformations from one coordinate reference system to another coordinate reference system.
SDO_CS.DETERMINE_DEFAULT_CHAIN	Returns the default chain of SRID values in transformations from one coordinate reference system to another coordinate reference system.
SDO_CS.FIND_GEOG_CRCS	Returns the SRID values of geodetic (geographic) coordinate reference systems that have the same well-known text (WKT) numeric values as the coordinate reference system with the specified reference SRID value.
SDO_CS.FIND_PROJ_CRCS	Returns the SRID values of geodetic (geographic) coordinate reference systems that have the same well-known text (WKT) numeric values as the coordinate reference system with the specified reference SRID value.
SDO_CS.FROM_OGC_SIMPLEFEATURE_SRS	Returns the SRID values of projected coordinate reference systems that have the same well-known text (WKT) numeric values as the coordinate reference system with the specified reference SRID value.
SDO_CS.MAP_EPSG_SRID_TO_ORACLE	Returns the EPSG SRID value corresponding to the specified Oracle Spatial SRID value.
SDO_CS.REVOKE_PREFERENCE_FOR_OP	Revokes a preference for an operation between a source coordinate system and a target coordinate system.
SDO_CS.TO_OGC_SIMPLEFEATURE_SRS	Converts a well-known text string from the Open Geospatial Consortium simple feature format that includes the TOWGS84 keyword to the format without the TOWGS84 keyword.

New SDO_GEOMETRY Methods and Constructors

The SDO_GEOMETRY object type has the following new methods (member functions): [Get_WKB](#), [Get_WKT](#), [ST_CoordDim](#), and [ST_IsValid](#). The SDO_GEOMETRY methods are described in [Section 2.3](#). (The methods were described in a separate chapter in the previous release of this guide.)

The SDO_GEOMETRY object type has new constructors for creating a geometry object from a well-known text (WKT) string in CLOB or VARCHAR2 format, or from a well-known binary (WKT) object in BLOB format. These constructors are described in [Section 2.4](#).

New min_resolution and max_resolution Keywords

The [SDO_FILTER](#), [SDO_RELATE](#), and [SDO_WITHIN_DISTANCE](#) operators accept the following new optional keywords in their last parameter:

- `min_resolution` includes only geometries for which at least one side of the geometry's MBR is equal to or greater than the specified value. For example, `min_resolution=10` includes only geometries for which the width or the height (or both) of the geometry's MBR is at least 10. (This keyword can be used to exclude geometries that are too small to be of interest.)

- `max_resolution` includes only geometries for which at least one side of the geometry's MBR is less than or equal to the specified value. For example, `max_resolution=10` includes only geometries for which the width or the height (or both) of the geometry's MBR is less than or equal to 10. (This keyword can be used to exclude geometries that are too large to be of interest.)

The Spatial operators are described in [Chapter 11](#).

New `sdo_dml_batch_size` Parameter

The new `sdo_dml_batch_size` parameter for the [CREATE INDEX](#) statement can affect the performance of applications that insert or update spatial data. This parameter specifies the number of index updates to be processed in each batch of updates after a commit operation. The default value is 1000. See the [CREATE INDEX](#) statement reference information and Usage Note in [Chapter 10](#) for details.

This parameter is part of a change for this release in how Spatial updates the spatial index when rows are inserted, updated, or deleted in the spatial table. In previous releases, Spatial updated the index after each DML (insert, update, or delete) operation on the table. In the current release, Spatial does not update the index until after the commit operation completes the transaction, and it updates the index in batches of operations determined by the `sdo_dml_batch_size` value. The result is faster processing of each DML operation and some extra time required after the commit operation, but a substantial overall performance improvement.

As a result of this change in spatial index update behavior, as well as other internal improvements, you are no longer encouraged to perform commit operations after each spatial table insert or update operation. Instead, for better performance (especially with large numbers of operations), perform the commit operation after a number of insert or update operations that is greater than the `sdo_dml_batch_size` value. There is no longer a risk of deadlocks or other locking problems if you delay performing the commit operation, even if multiple users in the same session are updating the spatial table.

New Geocoding Subprograms

The `SDO_GCDR` package (described in [Chapter 14](#)) contains the following new subprograms:

Subprogram	Description
SDO_GCDR.GEOCODE_ADDR	Geocodes an input address using attributes in an <code>SDO_GEO_ADDR</code> object, and returns the first matched address as an <code>SDO_GEO_ADDR</code> object.
SDO_GCDR.GEOCODE_ADDR_ALL	Geocodes an input address using attributes in an <code>SDO_GEO_ADDR</code> object, and returns matching addresses as an <code>SDO_ADDR_ARRAY</code> object.
SDO_GCDR.REVERSE_GEOCODE	Reverse geocodes a location, specified by its spatial geometry object and country, and returns an <code>SDO_GEO_ADDR</code> object.

New Utility Subprograms

The `SDO_UTIL` package (described in [Chapter 20](#)) contains the following new subprograms:

Subprogram	Description
SDO_UTIL.FROM_WKBGEOMETRY	Converts a geometry in the well-known binary (WKB) format to a Spatial geometry object.
SDO_UTIL.FROM_WKTGEOMETRY	Converts a geometry in the well-known text (WKT) format to a Spatial geometry object.
SDO_UTIL.RECTIFY_GEOMETRY	Fixes certain problems with the input geometry, and returns a valid geometry.
SDO_UTIL.TO_WKBGEOMETRY	Converts a Spatial geometry object to the well-known binary (WKB) format.
SDO_UTIL.TO_WKTGEOMETRY	Converts a Spatial geometry object to the well-known text (WKT) format.
SDO_UTIL.VALIDATE_WKBGEOMETRY	Validates the input geometry, which is in the standard well-known binary (WKB) format; returns the string <code>TRUE</code> if the geometry is valid or <code>FALSE</code> if the geometry is not valid.
SDO_UTIL.VALIDATE_WKTGEOMETRY	Validates the input geometry, which is of type <code>CLOB</code> or <code>VARCHAR2</code> and in the standard well-known text (WKT) format; returns the string <code>TRUE</code> if the geometry is valid or <code>FALSE</code> if the geometry is not valid.

U.S. National Grid Support

Spatial now supports the conversion of point locations between the U.S. National Grid format and the `SDO_GEOMETRY` point format. Support for the U.S. National Grid point representation format is described in [Section 6.10](#).

Unknown and NaC Coordinate Reference Systems

The unknown CRS (SRID 999999) and NaC (SRID 999998) coordinate reference systems have been added for Oracle internal use and other possible special uses. These coordinate reference systems are described in [Section 6.9.4](#).

Spatial Routing Engine

Spatial now includes a routing engine, which enables you to host an XML-based Web service that provides information about a single driving route or multiple routes. The Spatial routing engine is described in [Appendix C](#).

LRS_INTERSECTION Function

Effective with release 10.2.0.3, the new [SDO_LRS.LRS_INTERSECTION](#) function (described in [Chapter 16](#)) enables you to return an LRS geometry object that is the topological intersection (AND operation) of an LRS geometric segment and another geometry object.

Part I

Conceptual and Usage Information

This document has three parts:

- Part I provides conceptual and usage information about Oracle Spatial.
- [Part II](#) provides reference information about Oracle Spatial operators, functions, and procedures.
- [Part III](#) provides supplementary information (appendixes and a glossary).

Part I is organized for efficient learning about Oracle Spatial. It covers basic concepts and techniques first, and proceeds to more advanced material (such as coordinate systems, the linear referencing system, geocoding, and extending spatial indexing). Part I contains the following chapters:

- [Chapter 1, "Spatial Concepts"](#)
- [Chapter 2, "Spatial Data Types and Metadata"](#)
- [Chapter 3, "Loading Spatial Data"](#)
- [Chapter 4, "Indexing and Querying Spatial Data"](#)
- [Chapter 5, "Geocoding Address Data"](#)
- [Chapter 6, "Coordinate Systems \(Spatial Reference Systems\)"](#)
- [Chapter 7, "Linear Referencing System"](#)
- [Chapter 8, "Spatial Analysis and Mining"](#)
- [Chapter 9, "Extending Spatial Indexing Capabilities"](#)

Spatial Concepts

Oracle Spatial is an integrated set of functions and procedures that enables spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle database.

Spatial data represents the essential location characteristics of real or conceptual objects as those objects relate to the real or conceptual space in which they exist.

This chapter contains the following major sections:

- [Section 1.1, "What Is Oracle Spatial?"](#)
- [Section 1.2, "Object-Relational Model"](#)
- [Section 1.3, "Introduction to Spatial Data"](#)
- [Section 1.4, "Geometry Types"](#)
- [Section 1.5, "Data Model"](#)
- [Section 1.6, "Query Model"](#)
- [Section 1.7, "Indexing of Spatial Data"](#)
- [Section 1.8, "Spatial Relationships and Filtering"](#)
- [Section 1.9, "Spatial Operators, Procedures, and Functions"](#)
- [Section 1.10, "Spatial Aggregate Functions"](#)
- [Section 1.11, "Geocoding"](#)
- [Section 1.12, "Spatial Java Application Programming Interface"](#)
- [Section 1.13, "MDDATA Schema"](#)
- [Section 1.14, "Performance and Tuning Information"](#)
- [Section 1.15, "Open Geospatial Consortium \(OGC\) Conformance"](#)
- [Section 1.16, "Spatial Release \(Version\) Number"](#)
- [Section 1.17, "Spatial Application Hardware Requirement Considerations"](#)
- [Section 1.18, "Spatial Error Messages"](#)
- [Section 1.19, "Spatial Examples"](#)
- [Section 1.20, "README File for Spatial and Related Features"](#)

1.1 What Is Oracle Spatial?

Oracle Spatial, often referred to as Spatial, provides a SQL schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial features in an Oracle database. Spatial consists of the following:

- A schema (MDSYS) that prescribes the storage, syntax, and semantics of supported geometric data types
- A spatial indexing mechanism
- Operators, functions, and procedures for performing area-of-interest queries, spatial join queries, and other spatial analysis operations
- Functions and procedures for utility and tuning operations
- Topology data model for working with data about nodes, edges, and faces in a topology (described in *Oracle Spatial Topology and Network Data Models*).
- Network data model for representing capabilities or objects that are modeled as nodes and links in a network (described in *Oracle Spatial Topology and Network Data Models*).
- GeoRaster, a feature that lets you store, index, query, analyze, and deliver GeoRaster data, that is, raster image and gridded data and its associated metadata (described in *Oracle Spatial GeoRaster*).

The spatial component of a spatial feature is the geometric representation of its shape in some coordinate space. This is referred to as its **geometry**.

Caution: Do not modify any packages, tables, or other objects under the MDSYS schema. (The only exception is if you need to create a user-defined coordinate reference system, as explained in [Section 6.8](#).)

1.2 Object-Relational Model

Spatial supports the **object-relational** model for representing geometries. This model stores an entire geometry in the Oracle native spatial data type for vector data, SDO_GEOMETRY. An Oracle table can contain one or more SDO_GEOMETRY columns. The object-relational model corresponds to a "SQL with Geometry Types" implementation of spatial feature tables in the Open GIS ODBC/SQL specification for geospatial features.

Note: The relational geometry model of Oracle Spatial is no longer supported, effective with Oracle release 9.2. Only the object-relational model is supported.

The benefits provided by the object-relational model include:

- Support for many geometry types, including arcs, circles, compound polygons, compound line strings, and optimized rectangles
- Ease of use in creating and maintaining indexes and in performing spatial queries
- Index maintenance by the Oracle database
- Geometries modeled in a single column
- Optimal performance

1.3 Introduction to Spatial Data

Oracle Spatial is designed to make spatial data management easier and more natural to users of location-enabled applications and geographic information system (GIS) applications. Once spatial data is stored in an Oracle database, it can be easily manipulated, retrieved, and related to all other data stored in the database.

A common example of spatial data can be seen in a road map. A road map is a two-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. A road map is a visualization of geographic information. The location of cities, roads, and political boundaries that exist on the surface of the Earth are projected onto a two-dimensional display or piece of paper, preserving the relative positions and relative distances of the rendered objects.

The data that indicates the Earth location (such as longitude and latitude) of these rendered objects is the spatial data. When the map is rendered, this spatial data is used to project the locations of the objects on a two-dimensional piece of paper. A GIS is often used to store, retrieve, and render this Earth-relative spatial data.

Types of spatial data (other than GIS data) that can be stored using Spatial include data from computer-aided design (CAD) and computer-aided manufacturing (CAM) systems. Instead of operating on objects on a geographic scale, CAD/CAM systems work on a smaller scale, such as for an automobile engine or printed circuit boards.

The differences among these systems are in the size and precision of the data, not the data's complexity. The systems might all involve the same number of data points. On a geographic scale, the location of a bridge can vary by a few tenths of an inch without causing any noticeable problems to the road builders, whereas if the diameter of an engine's pistons is off by a few tenths of an inch, the engine will not run.

In addition, the complexity of data is independent of the absolute scale of the area being represented. For example, a printed circuit board is likely to have many thousands of objects etched on its surface, containing in its small area information that may be more complex than the details shown on a road builder's blueprints.

These applications all store, retrieve, update, or query some collection of features that have both nonspatial and spatial attributes. Examples of nonspatial attributes are name, soil_type, landuse_classification, and part_number. The spatial attribute is a coordinate geometry, or vector-based representation of the shape of the feature.

1.4 Geometry Types

A **geometry** is an ordered sequence of vertices that are connected by straight line segments or circular arcs. The semantics of the geometry are determined by its type. Spatial supports several primitive types, and geometries composed of collections of these types, including two-dimensional:

- Points and point clusters
- Line strings
- *n*-point polygons
- Arc line strings (All arcs are generated as circular arcs.)
- Arc polygons
- Compound polygons
- Compound line strings

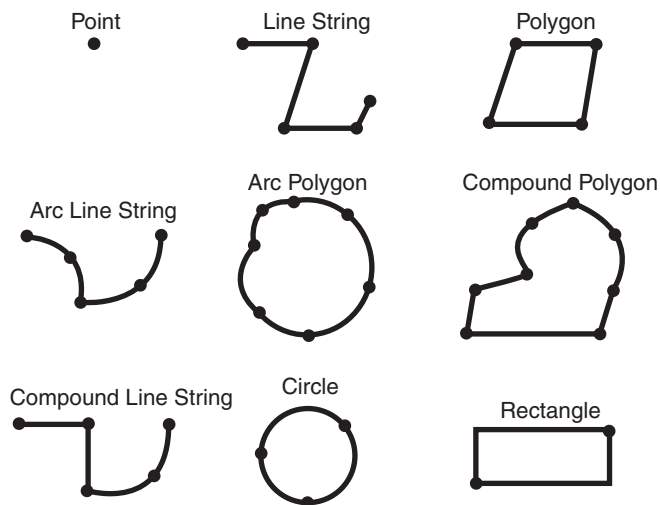
- Circles
- Optimized rectangles

Two-dimensional points are elements composed of two ordinates, X and Y, often corresponding to longitude and latitude. **Line strings** are composed of one or more pairs of points that define line segments. **Polygons** are composed of connected line strings that form a closed ring, and the area of the polygon is implied. For example, a point might represent a building location, a line string might represent a road or flight path, and a polygon might represent a state, city, zoning district, or city block.

Self-crossing polygons are not supported, although self-crossing line strings are supported. If a line string crosses itself, it does not become a polygon. A self-crossing line string does not have any implied area.

Figure 1–1 illustrates the geometric types.

Figure 1–1 Geometric Types



Spatial also supports the storage and indexing of three-dimensional and four-dimensional geometric types, where three or four coordinates are used to define each vertex of the object being defined. However, spatial functions (except for LRS functions and MBR-related functions) can work with only the first two dimensions, and all spatial operators except [SDO_FILTER](#) are disabled if the spatial index has been created on more than two dimensions.

1.5 Data Model

The Spatial data model is a hierarchical structure consisting of elements, geometries, and layers. Layers are composed of geometries, which in turn are made up of elements.

1.5.1 Element

An **element** is the basic building block of a geometry. The supported spatial element types are points, line strings, and polygons. For example, elements might model star constellations (point clusters), roads (line strings), and county boundaries (polygons). Each coordinate in an element is stored as an X,Y pair. The exterior ring and zero or more interior rings (holes) of a complex polygon are considered a single element.

Point data consists of one coordinate. **Line data** consists of two coordinates representing a line segment of the element. **Polygon data** consists of coordinate pair values, one vertex pair for each line segment of the polygon. Coordinates are defined in order around the polygon (counterclockwise for an exterior polygon ring, clockwise for an interior polygon ring).

1.5.2 Geometry

A **geometry** (or **geometry object**) is the representation of a spatial feature, modeled as an ordered set of primitive elements. A geometry can consist of a single element, which is an instance of one of the supported primitive types, or a homogeneous or heterogeneous collection of elements. A multipolygon, such as one used to represent a set of islands, is a homogeneous collection. A heterogeneous collection is one in which the elements are of different types, for example, a point and a polygon.

An example of a geometry might describe the buildable land in a town. This could be represented as a polygon with holes where water or zoning prevents construction.

1.5.3 Layer

A **layer** is a collection of geometries having the same attribute set. For example, one layer in a GIS might include topographical features, while another describes population density, and a third describes the network of roads and bridges in the area (lines and points). The geometries and associated spatial index for each layer are stored in the database in standard tables.

1.5.4 Coordinate System

A **coordinate system** (also called a *spatial reference system*) is a means of assigning coordinates to a location and establishing relationships between sets of such coordinates. It enables the interpretation of a set of coordinates as a representation of a position in a real world space.

Any spatial data has a coordinate system associated with it. The coordinate system can be *georeferenced* (related to a specific representation of the Earth) or not georeferenced (that is, Cartesian, and not related to a specific representation of the Earth). If the coordinate system is georeferenced, it has a default *unit of measurement* (such as meters) associated with it, but you can have Spatial automatically return results in another specified unit (such as miles). (For more information about unit of measurement support, see [Section 2.8](#).)

Before Oracle Spatial release 8.1.6, geometries (objects of type SDO_GEOMETRY) were stored as strings of coordinates without reference to any specific coordinate system. Spatial functions and operators always assumed a coordinate system that had the properties of an orthogonal Cartesian system, and sometimes did not provide correct results if Earth-based geometries were stored in longitude and latitude coordinates. With release 8.1.6, Spatial provided support for many different coordinate systems, and for converting data freely between different coordinate systems.

Spatial data can be associated with a Cartesian, geodetic (geographical), projected, or local coordinate system:

- **Cartesian coordinates** are coordinates that measure the position of a point from a defined origin along axes that are perpendicular in the represented two-dimensional or three-dimensional space.

If a coordinate system is not explicitly associated with a geometry, a Cartesian coordinate system is assumed.

- **Geodetic coordinates** (sometimes called *geographic coordinates*) are angular coordinates (longitude and latitude), closely related to spherical polar coordinates, and are defined relative to a particular Earth geodetic datum. (A geodetic datum is a means of representing the figure of the Earth and is the reference for the system of geodetic coordinates.)
- **Projected coordinates** are planar Cartesian coordinates that result from performing a mathematical mapping from a point on the Earth's surface to a plane. There are many such mathematical mappings, each used for a particular purpose.
- **Local coordinates** are Cartesian coordinates in a non-Earth (non-georeferenced) coordinate system. Local coordinate systems are often used for CAD applications and local surveys.

When performing operations on geometries, Spatial uses either a Cartesian or curvilinear computational model, as appropriate for the coordinate system associated with the spatial data.

For more information about coordinate system support in Spatial, including geodetic, projected, and local coordinates and coordinate system transformation, see [Chapter 6](#).

1.5.5 Tolerance

Tolerance is used to associate a level of precision with spatial data. **Tolerance** reflects the *distance that two points can be apart and still be considered the same* (for example, to accommodate rounding errors). The tolerance value must be a positive number greater than zero. The significance of the value depends on whether or not the spatial data is associated with a geodetic coordinate system. (Geodetic and other types of coordinate systems are described in [Section 1.5.4](#).)

- For geodetic data (such as data identified by longitude and latitude coordinates), the tolerance value is a number of meters. For example, a tolerance value of 100 indicates a tolerance of 100 meters. The tolerance value for geodetic data should not be smaller than 0.05 (5 centimeters), and in most cases it should be larger. Spatial uses 0.05 as the tolerance value for geodetic data if you specify a smaller value.
- For non-geodetic data, the tolerance value is a number of the units that are associated with the coordinate system associated with the data. For example, if the unit of measurement is miles, a tolerance value of 0.005 indicates a tolerance of 0.005 (that is, 1/200) mile (approximately 26 feet), and a tolerance value of 2 indicates a tolerance of 2 miles.

In both cases, the smaller the tolerance value, the more precision is to be associated with the data.

A tolerance value is specified in two cases:

- In the geometry metadata definition for a layer (see [Section 1.5.5.1](#))
- As an input parameter to certain functions (see [Section 1.5.5.2](#))

For additional information about tolerance with linear referencing system (LRS) data, see [Section 7.6](#).

1.5.5.1 Tolerance in the Geometry Metadata for a Layer

The dimensional information for a layer includes a tolerance value. Specifically, the DIMINFO column (described in [Section 2.6.3](#)) of the xxx_SDO_GEOM_METADATA

views includes an `SDO_TOLERANCE` value for each dimension, and the value should be the same for each dimension.

If a function accepts an optional `tolerance` parameter and this parameter is null or not specified, the `SDO_TOLERANCE` value of the layer is used. Using the non-geodetic data from the example in [Section 2.1](#), the actual distance between geometries `cola_b` and `cola_d` is 0.846049894. If a query uses the [SDO_GEOM.SDO_DISTANCE](#) function to return the distance between `cola_b` and `cola_d` and does not specify a `tolerance` parameter value, the result depends on the `SDO_TOLERANCE` value of the layer. For example:

- If the `SDO_TOLERANCE` value of the layer is 0.005, this query returns .846049894.
- If the `SDO_TOLERANCE` value of the layer is 0.5, this query returns 0.

The zero result occurs because Spatial first constructs an imaginary buffer of the tolerance value (0.5) around each geometry to be considered, and the buffers around `cola_b` and `cola_d` overlap in this case.

You can therefore take either of two approaches in selecting an `SDO_TOLERANCE` value for a layer:

- The value can reflect the desired level of precision in queries for distances between objects. For example, if two non-geodetic geometries 0.8 units apart should be considered as separated, specify a small `SDO_TOLERANCE` value such as 0.05 or smaller.
- The value can reflect the precision of the values associated with geometries in the layer. For example, if all geometries in a non-geodetic layer are defined using integers and if two objects 0.8 units apart should not be considered as separated, an `SDO_TOLERANCE` value of 0.5 is appropriate. To have greater precision in any query, you must override the default by specifying the `tolerance` parameter.

With non-geodetic data, the guideline to follow for most instances of the second case (precision of the values of the geometries in the layer) is: take the highest level of precision in the geometry definitions, and use .5 at the next level as the `SDO_TOLERANCE` value. For example, if geometries are defined using integers (as in the simplified example in [Section 2.1](#)), the appropriate value is 0.5; however, if geometries are defined using numbers up to four decimal positions (for example, 31.2587), the appropriate value is 0.00005.

Note: This guideline should not be used if the geometries include any polygons that are so narrow at any point that the distance between facing sides is less than the proposed tolerance value. Be sure that the tolerance value is less than the shortest distance between any two sides in any polygon.

Moreover, if you encounter "invalid geometry" errors with inserted or updated geometries, and if the geometries are in fact valid, consider increasing the precision of the tolerance value (for example, changing 0.00005 to 0.000005).

1.5.5.2 Tolerance as an Input Parameter

Many Spatial functions accept a `tolerance` parameter, which (if specified) overrides the default tolerance value for the layer (explained in [Section 1.5.5.1](#)). If the distance between two points is less than or equal to the tolerance value, Spatial considers the two points to be a single point. Thus, tolerance is usually a reflection of how accurate or precise users perceive their spatial data to be.

For example, assume that you want to know which restaurants are within 5 kilometers of your house. Assume also that Maria's Pizzeria is 5.1 kilometers from your house. If the spatial data has a geodetic coordinate system and if you ask, *Find all restaurants within 5 kilometers and use a tolerance of 100* (or greater, such as 500), Maria's Pizzeria will be included, because 5.1 kilometers (5100 meters) is within 100 meters of 5 kilometers (5000 meters). However, if you specify a tolerance less than 100 (such as 50), Maria's Pizzeria will not be included.

Tolerance values for Spatial functions are typically very small, although the best value in each case depends on the kinds of applications that use or will use the data.

1.6 Query Model

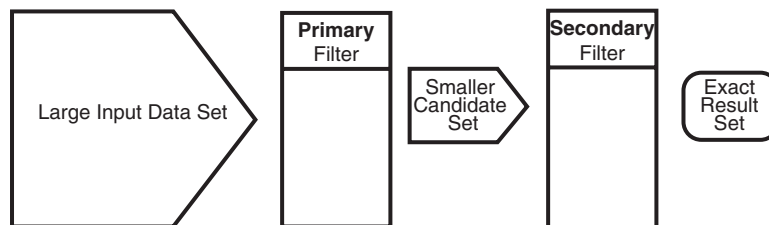
Spatial uses a **two-tier query model** to resolve spatial queries and spatial joins. The term is used to indicate that two distinct operations are performed to resolve queries. The output of the two combined operations yields the exact result set.

The two operations are referred to as *primary* and *secondary* filter operations.

- The **primary filter** permits fast selection of candidate records to pass along to the secondary filter. The primary filter compares geometry approximations to reduce computation complexity and is considered a lower-cost filter. Because the primary filter compares geometric approximations, it returns a superset of the exact result set.
- The **secondary filter** applies exact computations to geometries that result from the primary filter. The secondary filter yields an accurate answer to a spatial query. The secondary filter operation is computationally expensive, but it is only applied to the primary filter results, not the entire data set.

Figure 1–2 illustrates the relationship between the primary and secondary filters.

Figure 1–2 Query Model



As shown in Figure 1–2, the primary filter operation on a large input data set produces a smaller candidate set, which contains at least the exact result set and may contain more records. The secondary filter operation on the smaller candidate set produces the exact result set.

Spatial uses a spatial index to implement the primary filter. Spatial does not require the use of both the primary and secondary filters. In some cases, just using the primary filter is sufficient. For example, a *zoom* feature in a mapping application queries for data that has any interaction with a rectangle representing visible boundaries. The primary filter very quickly returns a superset of the query. The mapping application can then apply clipping routines to display the target area.

The purpose of the primary filter is to quickly create a subset of the data and reduce the processing burden on the secondary filter. The primary filter, therefore, should be

as efficient (that is, selective yet fast) as possible. This is determined by the characteristics of the spatial index on the data.

For more information about querying spatial data, see [Section 4.2](#).

1.7 Indexing of Spatial Data

The introduction of spatial indexing capabilities into the Oracle database engine is a key feature of the Spatial product. A spatial index, like any other index, provides a mechanism to limit searches, but in this case the mechanism is based on spatial criteria such as intersection and containment. A spatial index is needed to:

- Find objects within an indexed data space that interact with a given point or area of interest (window query)
- Find pairs of objects from within two indexed data spaces that interact spatially with each other (spatial join)

A spatial index is considered a logical index. The entries in the spatial index are dependent on the location of the geometries in a coordinate space, but the index values are in a different domain. Index entries may be ordered using a linearly ordered domain, and the coordinates for a geometry may be pairs of integer, floating-point, or double-precision numbers.

Oracle Spatial lets you use R-tree indexing (the default) or quadtree indexing, or both. However, the use of quadtree indexes is discouraged, and you are strongly encouraged to use R-tree indexing. Significant performance improvements have been made to spatial R-tree indexing for this release. Quadtree indexing is a deprecated feature of Spatial. Almost all information about quadtree indexing has been removed from this guide and placed in a separate guide, *Oracle Spatial Quadtree Indexing*, which is available only through the Oracle Technology Network.

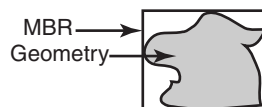
Testing of spatial indexes with many workloads and operators is ongoing, and results and recommendations will be documented as they become available.

The following sections explain the concepts and options associated with R-tree indexing.

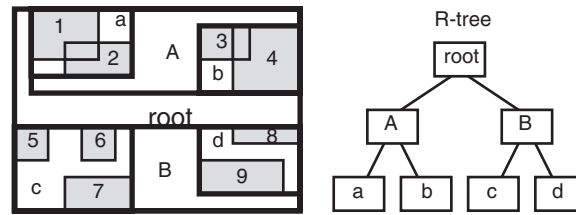
1.7.1 R-Tree Indexing

A spatial R-tree index can index spatial data of up to four dimensions. An R-tree index approximates each geometry by a single rectangle that minimally encloses the geometry (called the minimum bounding rectangle, or MBR), as shown in [Figure 1–3](#).

Figure 1–3 MBR Enclosing a Geometry



For a layer of geometries, an R-tree index consists of a hierarchical index on the MBRs of the geometries in the layer, as shown in [Figure 1–4](#).

Figure 1–4 R-Tree Hierarchical Index on MBRs

In [Figure 1–4](#):

- 1 through 9 are geometries in a layer.
- *a*, *b*, *c*, and *d* are the leaf nodes of the R-tree index, and contain minimum bounding rectangles of geometries, along with pointers to the geometries. For example, *a* contains the MBR of geometries 1 and 2, *b* contains the MBR of geometries 3 and 4, and so on.
- *A* contains the MBR of *a* and *b*, and *B* contains the MBR of *c* and *d*.
- The root contains the MBR of *A* and *B* (that is, the entire area shown).

An R-tree index is stored in the spatial index table (SDO_INDEX_TABLE in the USER_SDO_INDEX_METADATA view, described in [Section 2.7](#)). The R-tree index also maintains a sequence object (SDO_RTREE_SEQ_NAME in the USER_SDO_INDEX_METADATA view) to ensure that simultaneous updates by concurrent users can be made to the index.

1.7.2 R-Tree Quality

A substantial number of insert and delete operations affecting an R-tree index may degrade the quality of the R-tree structure, which may adversely affect query performance.

The R-tree is a hierarchical tree structure with nodes at different heights of the tree. The performance of an R-tree index structure for queries is roughly proportional to the area and perimeter of the index nodes of the R-tree. The area covered at level 0 represents the area occupied by the minimum bounding rectangles of the data geometries, the area at level 1 indicates the area covered by leaf-level R-tree nodes, and so on. The original ratio of the area at the root (topmost level) to the area at level 0 can change over time based on updates to the table; and if there is a degradation in that ratio (that is, if it increases significantly), rebuilding the index may help the performance of queries.

If the performance of [SDO_FILTER](#) operations has degraded, and if there have been a large number of insert, update, or delete operations affecting geometries, the performance degradation may be due to a degradation in the quality of the associated R-tree index. You can check for degradation of index quality by using the [SDO_TUNE.QUALITY_DEGRADATION](#) function (described in [Chapter 19](#)): if the function returns a number greater than 2, consider rebuilding the index. Note, however, that the R-tree index quality degradation number may not be significant in terms of overall query performance due to Oracle caching strategies and other significant Oracle capabilities, such as table pinning, which can essentially remove I/O overhead from R-tree index queries.

To rebuild an R-tree index, use the [ALTER INDEX REBUILD](#) statement, which is described in [Chapter 10](#).

1.8 Spatial Relationships and Filtering

Spatial uses secondary filters to determine the spatial relationship between entities in the database. The spatial relationship is based on geometry locations. The most common spatial relationships are based on topology and distance. For example, the *boundary* of an area consists of a set of curves that separates the area from the rest of the coordinate space. The *interior* of an area consists of all points in the area that are not on its boundary. Given this, two areas are said to be adjacent if they share part of a boundary but do not share any points in their interior.

The distance between two spatial objects is the minimum distance between any points in them. Two objects are said to be *within a given distance* of one another if their distance is less than the given distance.

To determine spatial relationships, Spatial has several secondary filter methods:

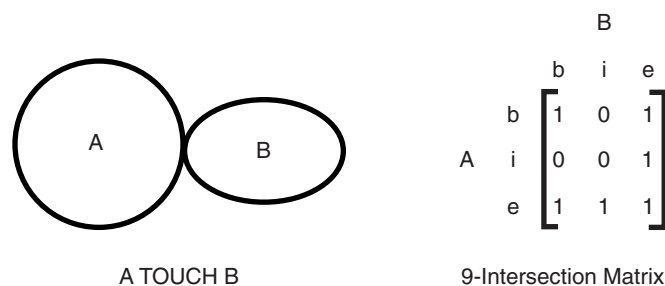
- The `SDO_RELATE` operator evaluates topological criteria.
- The `SDO_WITHIN_DISTANCE` operator determines if two spatial objects are within a specified distance of each other.
- The `SDO_NN` operator identifies the nearest neighbors for a spatial object.

The syntax of these operators is given in [Chapter 11](#).

The `SDO_RELATE` operator implements a nine-intersection model for categorizing binary topological relationships between points, lines, and polygons. Each spatial object has an interior, a boundary, and an exterior. The boundary consists of points or lines that separate the interior from the exterior. The boundary of a line string consists of its end points; however, if the end points overlap (that is, if they are the same point), the line string has no boundary. The boundaries of a multiline string are the end points of each of the component line strings; however, if the end points overlap, only the end points that overlap an odd number of times are boundaries. The boundary of a polygon is the line that describes its perimeter. The interior consists of points that are in the object but not on its boundary, and the exterior consists of those points that are not in the object.

Given that an object A has three components (a boundary A_b , an interior A_i , and an exterior A_e), any pair of objects has nine possible interactions between their components. Pairs of components have an empty (0) or not empty (1) set intersection. The set of interactions between two geometries is represented by a nine-intersection matrix that specifies which pairs of components intersect and which do not. [Figure 1–5](#) shows the nine-intersection matrix for two polygons that are adjacent to one another. This matrix yields the following bit mask, generated in row-major form: "101001111".

Figure 1–5 The Nine-Intersection Model

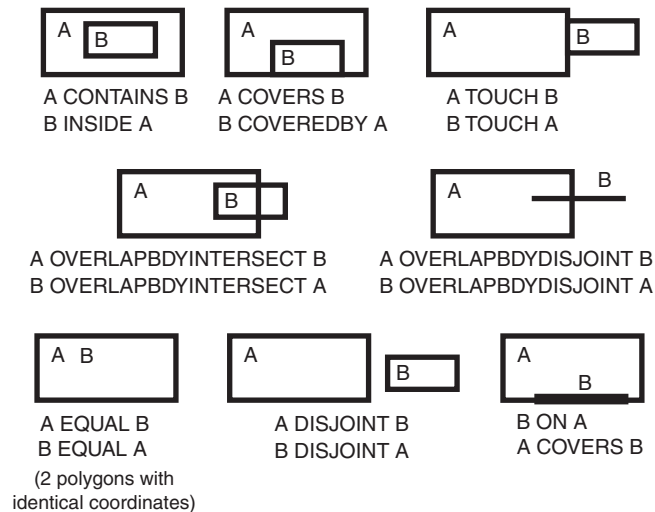


Some of the topological relationships identified in the seminal work by Professor Max Egenhofer (University of Maine, Orono) and colleagues have names associated with them. Spatial uses the following names:

- DISJOINT -- The boundaries and interiors do not intersect.
- TOUCH -- The boundaries intersect but the interiors do not intersect.
- OVERLAPBDYDISJOINT -- The interior of one object intersects the boundary and interior of the other object, but the two boundaries do not intersect. This relationship occurs, for example, when a line originates outside a polygon and ends inside that polygon.
- OVERLAPBDYINTERSECT -- The boundaries and interiors of the two objects intersect.
- EQUAL -- The two objects have the same boundary and interior.
- CONTAINS -- The interior and boundary of one object is completely contained in the interior of the other object.
- COVERS -- The interior of one object is completely contained in the interior or the boundary of the other object and their boundaries intersect.
- INSIDE -- The opposite of CONTAINS. A INSIDE B implies B CONTAINS A.
- COVEREDBY -- The opposite of COVERS. A COVEREDBY B implies B COVERS A.
- ON -- The interior and boundary of one object is on the boundary of the other object (and the second object covers the first object). This relationship occurs, for example, when a line is on the boundary of a polygon.
- ANYINTERACT -- The objects are non-disjoint.

Figure 1–6 illustrates these topological relationships.

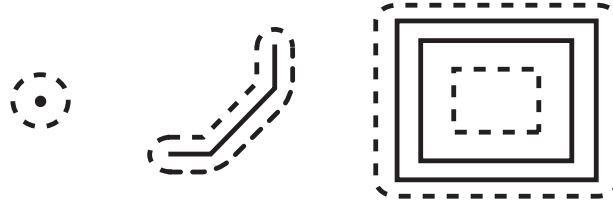
Figure 1–6 Topological Relationships



The `SDO_WITHIN_DISTANCE` operator determines if two spatial objects, A and B, are within a specified distance of one another. This operator first constructs a distance buffer, D_b , around the reference object B. It then checks that A and D_b are non-disjoint.

The distance buffer of an object consists of all points within the given distance from that object. [Figure 1-7](#) shows the distance buffers for a point, a line, and a polygon.

Figure 1-7 Distance Buffers for Points, Lines, and Polygons



In the point, line, and polygon geometries shown in [Figure 1-7](#):

- The dashed lines represent distance buffers. Notice how the buffer is rounded near the corners of the objects.
- The geometry on the right is a polygon with a hole: the large rectangle is the exterior polygon ring and the small rectangle is the interior polygon ring (the hole). The dashed line outside the large rectangle is the buffer for the exterior ring, and the dashed line inside the small rectangle is the buffer for the interior ring.

The [SDO_NN](#) operator returns a specified number of objects from a geometry column that are closest to a specified geometry (for example, the five closest restaurants to a city park). In determining how close two geometry objects are, the shortest possible distance between any two points on the surface of each object is used.

1.9 Spatial Operators, Procedures, and Functions

The Spatial PL/SQL application programming interface (API) includes several operators and many procedures and functions.

Spatial operators, such as [SDO_FILTER](#) and [SDO_RELATE](#), provide optimum performance because they use the spatial index. (Spatial operators require that the geometry column in the first parameter have a spatial index defined on it.) Spatial operators must be used in the WHERE clause of a query. The first parameter of any operator specifies the geometry column to be searched, and the second parameter specifies a query window. If the query window does not have the same coordinate system as the geometry column, Spatial performs an implicit coordinate system transformation. For detailed information about the spatial operators, see [Chapter 11](#).

Spatial procedures and functions are provided as subprograms in PL/SQL packages, such as [SDO_GEOM](#), [SDO_CS](#), and [SDO_LRS](#). These subprograms do not require that a spatial index be defined, and they do not use a spatial index if it is defined. These subprograms can be used in the WHERE clause or in a subquery. If two geometries are input parameters to a Spatial procedure or function, both must have the same coordinate system.

The following performance-related guidelines apply to the use of spatial operators, procedures, and functions:

- If an operator and a procedure or function perform comparable operations, and if the operator satisfies your requirements, use the operator. For example, unless you need to do otherwise, use [SDO_RELATE](#) instead of [SDO_GEOM.RELATE](#), and use [SDO_WITHIN_DISTANCE](#) instead of [SDO_GEOM.WITHIN_DISTANCE](#).

- With operators, always specify TRUE in uppercase. That is, specify = 'TRUE', and do not specify <> 'FALSE' or = 'true'.
- With operators, use the /*+ ORDERED */ optimizer hint if the query window comes from a table. (You must use this hint if multiple windows come from a table.) See the Usage Notes and Examples for specific operators for more information.

For information about using operators with topologies, see *Oracle Spatial Topology and Network Data Models*.

1.10 Spatial Aggregate Functions

SQL has long had aggregate functions, which are used to aggregate the results of a SQL query. The following example uses the SUM aggregate function to aggregate employee salaries by department:

```
SELECT SUM(salary), dept
       FROM employees
       GROUP BY dept;
```

Oracle Spatial aggregate functions aggregate the results of SQL queries involving geometry objects. Spatial aggregate functions return a geometry object of type SDO_GEOMETRY. For example, the following statement returns the minimum bounding rectangle of all geometries in a table (using the definitions and data from [Section 2.1](#)):

```
SELECT SDO_AGGR_MBR(shape) FROM cola_markets;
```

The following example returns the union of all geometries except cola_d:

```
SELECT SDO_AGGR_UNION(SDOAGGRTYPE(c.shape, 0.005))
       FROM cola_markets c WHERE c.name < 'cola_d';
```

All geometries used with spatial aggregate functions must be defined using 4-digit SDO_GTYPE values (that is, must be in the format used by Oracle Spatial release 8.1.6 or later). For information about SDO_GTYPE values, see [Section 2.2.1](#).

For reference information about the spatial aggregate functions and examples of their use, see [Chapter 12](#).

1.10.1 SDOAGGRTYPE Object Type

Many spatial aggregate functions accept an input parameter of type SDOAGGRTYPE. Oracle Spatial defines the object type SDOAGGRTYPE as:

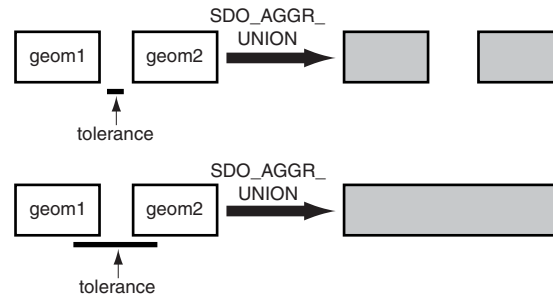
```
CREATE TYPE sdoaggrtype AS OBJECT (
  geometry SDO_GEOMETRY,
  tolerance NUMBER);
```

Note: Do not use SDOAGGRTYPE as the data type for a column in a table. Use this type only in calls to spatial aggregate functions.

The tolerance value in the SDOAGGRTYPE definition should be the same as the SDO_TOLERANCE value specified in the DIMINFO column in the xxx_SDO_GEOM_METADATA views for the geometries, unless you have a specific reason for wanting a different value. For more information about tolerance, see [Section 1.5.5](#); for information about the xxx_SDO_GEOM_METADATA views, see [Section 2.6](#).

The tolerance value in the SDOAGGRTYPE definition can affect the result of a spatial aggregate function. [Figure 1–8](#) shows a spatial aggregate union (SDO_AGGR_UNION) operation of two geometries using two different tolerance values: one smaller and one larger than the distance between the geometries.

Figure 1–8 Tolerance in an Aggregate Union Operation



In the first aggregate union operation in [Figure 1–8](#), where the tolerance is less than the distance between the rectangles, the result is a compound geometry consisting of two rectangles. In the second aggregate union operation, where the tolerance is greater than the distance between the rectangles, the result is a single geometry.

1.11 Geocoding

Geocoding is the process of converting tables of address data into standardized address, location, and possibly other data. The result of a geocoding operation includes the pair of longitude and latitude coordinates that correspond with the input address or location. For example, if the input address is *22 Monument Square, Concord, MA 01742*, the longitude and latitude coordinates in the result of the geocoding operation may be (depending on the geocoding data provider) *-71.34937* and *42.46101*, respectively.

Given a geocoded address, you can perform proximity or location queries using a spatial engine, such as Oracle Spatial, or demographic analysis using tools and data from Oracle’s business partners. In addition, you can use geocoded data with other spatial data such as block group, postal code, and county code for association with demographic information. Results of analyses or queries can be presented as maps, in addition to tabular formats, using third-party software integrated with Oracle Spatial.

For conceptual and usage information about the geocoding capabilities of Oracle Spatial, see [Chapter 5](#). For reference information about the MDSYS.SDO_GCDR PL/SQL package, see [Chapter 14](#).

1.12 Spatial Java Application Programming Interface

Oracle Spatial provides a Java application programming interface (API) that includes the following packages:

- `oracle.spatial.geometry` provides support for the Spatial SQL SDO_GEOMETRY data type, which is documented in this guide.
- `oracle.spatial.network` provides support for the Oracle Spatial network data model, which is documented in *Oracle Spatial Topology and Network Data Models*.

- `oracle.spatial.topo` provides support for the Oracle Spatial topology data model, which is documented in *Oracle Spatial Topology and Network Data Models*.
- `oracle.spatial.util` provides classes that perform miscellaneous operations.

For detailed reference information about the classes and interfaces in these packages, see *Oracle Spatial Java API Reference (Javadoc)*.

1.13 MDDATA Schema

Effective with Oracle Database 10g, Spatial creates a user and schema named MDDATA, using the following internal SQL statements:

```
CREATE USER mddata IDENTIFIED BY mddata;  
GRANT connect, resource TO mddata;  
ALTER USER mddata ACCOUNT LOCK;
```

You should use the MDDATA schema for storing data used by geocoding and routing applications. This is the default schema for Oracle software that accesses geocoding and routing data.

1.14 Performance and Tuning Information

Many factors can affect the performance of Oracle Spatial applications, such as the use of optimizer hints to influence the plan for query execution. This guide contains some information about performance and tuning where it is relevant to a particular topic. For example, [Section 1.7.2](#) discusses R-tree quality and its possible effect on query performance, and [Section 1.9](#) explains why spatial operators provide better performance than procedures and functions.

In addition, more Spatial performance and tuning information is available in one or more white papers through the Oracle Technology Network (OTN). That information is often more detailed than what is in this guide, and it is periodically updated as a result of internal testing and consultations with Spatial users. To find that information on the OTN, go to

<http://www.oracle.com/technology/products/spatial/>

Look for material relevant to Spatial performance and tuning.

1.15 Open Geospatial Consortium (OGC) Conformance

Oracle Spatial is conformant with Open Geospatial Consortium (OGC) Simple Features Specification 1.1.1 (Document 99-049), starting with Oracle Database release 10g (version 10.1.0.4). Conformance with the SQL92 with Geometry Types Implementation means that Oracle Spatial supports all the types, functions, and language constructs detailed in Section 3.2 of the specification.

1.16 Spatial Release (Version) Number

To check which release of Spatial you are running, use the `SDO_VERSION` function. For example:

```
SELECT SDO_VERSION FROM DUAL;
```

```
SDO_VERSION
```

```
-----  
10.2.0.0.0
```

1.17 Spatial Application Hardware Requirement Considerations

This section discusses some general guidelines that affect the amount of disk storage space and CPU power needed for applications that use Oracle Spatial. These guidelines are intended to supplement, not replace, any other guidelines you use for general application sizing.

The following characteristics of spatial applications can affect the need for storage space and CPU power:

- **Data volumes:** The amount of storage space needed for spatial objects depends on their complexity (precision of representation and number of points for each object). For example, storing one million point objects takes less space than storing one million road segments or land parcels. Complex natural features such as coastlines, seismic fault lines, rivers, and land types can require significant storage space if they are stored at a high precision.
- **Query complexity:** The CPU requirements for simple mapping queries, such as *Select all features in this rectangle*, are lower than for more complex queries, such as *Find all seismic fault lines that cross this coastline*.

1.18 Spatial Error Messages

Spatial error message numbers are in the range of 13000 to 13499. The messages are documented in *Oracle Database Error Messages*.

Oracle error message documentation is only available in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

1.19 Spatial Examples

Oracle Spatial provides examples that you can use to reinforce your learning and to create models for coding certain operations. If you installed the demo files from the Companion CD, several examples are provided in the following directory:

```
$ORACLE_HOME/md/demos/examples
```

The following files in that directory are helpful for applications that use the Oracle Call Interface (OCI):

- `readgeom.c` and `readgeom.h`
- `writegeom.c` and `writegeom.h`

This guide also includes many examples in SQL and PL/SQL. One or more examples are usually provided with the reference information for each function or procedure, and several simplified examples are provided that illustrate table and index creation, combinations of functions and procedures, and advanced features:

- Inserting, indexing, and querying spatial data ([Section 2.1](#))
- Coordinate systems (spatial reference systems) ([Section 6.11](#))
- Linear referencing system (LRS) ([Section 7.7](#))
- SDO_GEOMETRY objects in function-based indexes ([Section 9.2](#))
- Complex queries ([Appendix D](#))

1.20 README File for Spatial and Related Features

A `README.txt` file supplements the information in the following manuals: *Oracle Spatial User's Guide and Reference* (this manual), *Oracle Spatial GeoRaster*, and *Oracle Spatial Topology and Network Data Models*. This file is located at:

```
$ORACLE_HOME/md/doc/README.txt
```

Spatial Data Types and Metadata

Oracle Spatial consists of a set of object data types, type methods, and operators, functions, and procedures that use these types. A geometry is stored as an object, in a single row, in a column of type `SDO_GEOMETRY`. Spatial index creation and maintenance is done using basic DDL (`CREATE`, `ALTER`, `DROP`) and DML (`INSERT`, `UPDATE`, `DELETE`) statements.

This chapter starts with a simple example that inserts, indexes, and queries spatial data. You may find it helpful to read this example quickly before you examine the detailed data type and metadata information later in the chapter.

This chapter contains the following major sections:

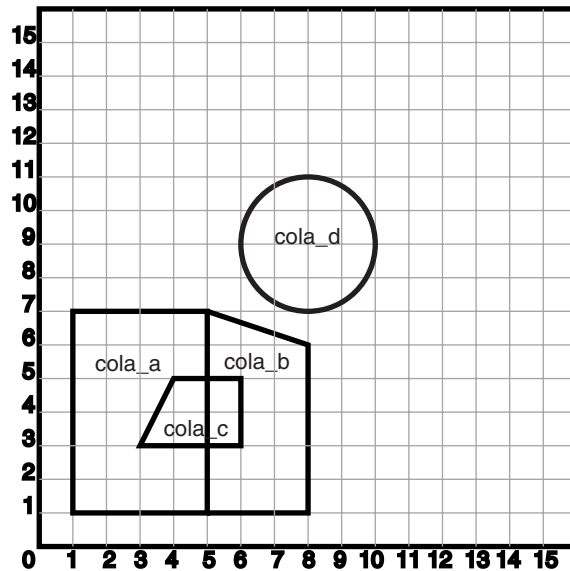
- [Section 2.1, "Simple Example: Inserting, Indexing, and Querying Spatial Data"](#)
- [Section 2.2, "SDO_GEOMETRY Object Type"](#)
- [Section 2.3, "SDO_GEOMETRY Methods"](#)
- [Section 2.4, "SDO_GEOMETRY Constructors"](#)
- [Section 2.5, "Geometry Examples"](#)
- [Section 2.6, "Geometry Metadata Views"](#)
- [Section 2.7, "Spatial Index-Related Structures"](#)
- [Section 2.8, "Unit of Measurement Support"](#)

2.1 Simple Example: Inserting, Indexing, and Querying Spatial Data

This section presents a simple example of creating a spatial table, inserting data, creating the spatial index, and performing spatial queries. It refers to concepts that were explained in [Chapter 1](#) and that will be explained in other sections of this chapter.

The scenario is a soft drink manufacturer that has identified geographical areas of marketing interest for several products (colas). The colas could be those produced by the company or by its competitors, or some combination. Each area of interest could represent any user-defined criterion: for example, an area where that cola has the majority market share, or where the cola is under competitive pressure, or where the cola is believed to have significant growth potential. Each area could be a neighborhood in a city, or a part of a state, province, or country.

[Figure 2–1](#) shows the areas of interest for four colas.

Figure 2–1 Areas of Interest for the Simple Example

[Example 2–1](#) performs the following operations:

- Creates a table (COLA_MARKETS) to hold the spatial data
- Inserts rows for four areas of interest (cola_a, cola_b, cola_c, cola_d)
- Updates the USER_SDO_GEOM_METADATA view to reflect the dimensional information for the areas
- Creates a spatial index (COLA_SPATIAL_IDX)
- Performs some spatial queries

Many concepts and techniques in [Example 2–1](#) are explained in detail in other sections of this chapter.

Example 2–1 Simple Example: Inserting, Indexing, and Querying Spatial Data

```
-- Create a table for cola (soft drink) markets in a
-- given geography (such as city or state).
-- Each row will be an area of interest for a specific
-- cola (for example, where the cola is most preferred
-- by residents, where the manufacturer believes the
-- cola has growth potential, and so on).
-- (For restrictions on spatial table and column names, see
-- Section 2.6.1 and Section 2.6.2.)
```

```
CREATE TABLE cola_markets (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape SDO_GEOMETRY);
```

```
-- The next INSERT statement creates an area of interest for
-- Cola A. This area happens to be a rectangle.
-- The area could represent any user-defined criterion: for
-- example, where Cola A is the preferred drink, where
-- Cola A is under competitive pressure, where Cola A
-- has strong growth potential, and so on.
```



```

INSERT INTO cola_markets VALUES(
  1,
  'cola_a',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
    SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
      -- define rectangle (lower left and upper right) with
      -- Cartesian-coordinate data
  )
);

-- The next two INSERT statements create areas of interest for
-- Cola B and Cola C. These areas are simple polygons (but not
-- rectangles).

INSERT INTO cola_markets VALUES(
  2,
  'cola_b',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), -- one polygon (exterior polygon ring)
    SDO_ORDINATE_ARRAY(5,1, 8,1, 8,6, 5,7, 5,1)
  )
);

INSERT INTO cola_markets VALUES(
  3,
  'cola_c',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), -- one polygon (exterior polygon ring)
    SDO_ORDINATE_ARRAY(3,3, 6,3, 6,5, 4,5, 3,3)
  )
);

-- Now insert an area of interest for Cola D. This is a
-- circle with a radius of 2. It is completely outside the
-- first three areas of interest.

INSERT INTO cola_markets VALUES(
  4,
  'cola_d',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,4), -- one circle
    SDO_ORDINATE_ARRAY(8,7, 10,9, 8,11)
  )
);

-----
-- UPDATE METADATA VIEW --

```

```

-----
-- Update the USER_SDO_GEOM_METADATA view. This is required
-- before the Spatial index can be created. Do this only once for each
-- layer (that is, table-column combination; here: COLA_MARKETS and SHAPE).

INSERT INTO user_sdo_geom_metadata
  (TABLE_NAME,
   COLUMN_NAME,
   DIMINFO,
   SRID)
VALUES (
  'cola_markets',
  'shape',
  SDO_DIM_ARRAY( -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
  ),
  NULL -- SRID
);

-----
-- CREATE THE SPATIAL INDEX --
-----

CREATE INDEX cola_spatial_idx
  ON cola_markets(shape)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;
-- Preceding statement created an R-tree index.

-----
-- PERFORM SOME SPATIAL QUERIES --
-----

-- Return the topological intersection of two geometries.
SELECT SDO_GEOM.SDO_INTERSECTION(c_a.shape, c_c.shape, 0.005)
  FROM cola_markets c_a, cola_markets c_c
  WHERE c_a.name = 'cola_a' AND c_c.name = 'cola_c';

-- Do two geometries have any spatial relationship?
SELECT SDO_GEOM.RELATE(c_b.shape, 'anyinteract', c_d.shape, 0.005)
  FROM cola_markets c_b, cola_markets c_d
  WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';

-- Return the areas of all cola markets.
SELECT name, SDO_GEOM.SDO_AREA(shape, 0.005) FROM cola_markets;

-- Return the area of just cola_a.
SELECT c.name, SDO_GEOM.SDO_AREA(c.shape, 0.005) FROM cola_markets c
  WHERE c.name = 'cola_a';

-- Return the distance between two geometries.
SELECT SDO_GEOM.SDO_DISTANCE(c_b.shape, c_d.shape, 0.005)
  FROM cola_markets c_b, cola_markets c_d
  WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';

-- Is a geometry valid?
SELECT c.name, SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(c.shape, 0.005)
  FROM cola_markets c WHERE c.name = 'cola_c';

-- Is a layer valid? (First, create the results table.)
CREATE TABLE val_results (sdo_rowid ROWID, result VARCHAR2(2000));
CALL SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT('COLA_MARKETS', 'SHAPE',

```

```
'VAL_RESULTS', 2);
SELECT * from val_results;
```

2.2 SDO_GEOMETRY Object Type

With Spatial, the geometric description of a spatial object is stored in a single row, in a single column of object type SDO_GEOMETRY in a user-defined table. Any table that has a column of type SDO_GEOMETRY must have another column, or set of columns, that defines a unique primary key for that table. Tables of this sort are sometimes referred to as spatial tables or spatial geometry tables.

Oracle Spatial defines the object type SDO_GEOMETRY as:

```
CREATE TYPE sdo_geometry AS OBJECT (
  SDO_GTYPE NUMBER,
  SDO_SRID NUMBER,
  SDO_POINT SDO_POINT_TYPE,
  SDO_ELEM_INFO SDO_ELEM_INFO_ARRAY,
  SDO_ORDINATES SDO_ORDINATE_ARRAY);
```

Oracle Spatial also defines the SDO_POINT_TYPE, SDO_ELEM_INFO_ARRAY, and SDO_ORDINATE_ARRAY types, which are used in the SDO_GEOMETRY type definition, as follows:

```
CREATE TYPE sdo_point_type AS OBJECT (
  X NUMBER,
  Y NUMBER,
  Z NUMBER);
CREATE TYPE sdo_elem_info_array AS VARRAY (1048576) of NUMBER;
CREATE TYPE sdo_ordinate_array AS VARRAY (1048576) of NUMBER;
```

Because the maximum SDO_ORDINATE_ARRAY size is 1,048,576 numbers, the maximum number of vertices in an SDO_GEOMETRY object depends on the number of dimensions per vertex: 524,288 for two dimensions, 349,525 for three dimensions, and 262,144 for four dimensions.

The sections that follow describe the semantics of each SDO_GEOMETRY attribute, and then describe some usage considerations ([Section 2.2.6](#)).

The SDO_GEOMETRY object type has methods that provide convenient access to some of the attributes. These methods are described in [Section 2.3](#).

Some Spatial data types are described in locations other than this section:

- [Section 5.2](#) describes data types for geocoding.
- *Oracle Spatial GeoRaster* describes data types for Oracle Spatial GeoRaster.
- *Oracle Spatial Topology and Network Data Models* describes data types for the Oracle Spatial topology data model.

2.2.1 SDO_GTYPE

The SDO_GTYPE attribute indicates the type of the geometry. Valid geometry types correspond to those specified in the *Geometry Object Model for the OGIS Simple Features for SQL* specification (with the exception of Surfaces). The numeric values differ from those given in the OGIS specification, but there is a direct correspondence between the names and semantics where applicable.

The SDO_GTYPE value is 4 digits in the format *dltt*, where:

- *d* identifies the number of dimensions (2, 3, or 4)

- *l* identifies the linear referencing measure dimension for a three-dimensional linear referencing system (LRS) geometry, that is, which dimension (3 or 4) contains the measure value. For a non-LRS geometry, or to accept the Spatial default of the last dimension as the measure for an LRS geometry, specify 0. For information about the linear referencing system (LRS), see [Chapter 7](#).
- *tt* identifies the geometry type (00 through 07, with 08 through 99 reserved for future use).

[Table 2–1](#) shows the valid SDO_GTYPE values. The Geometry Type and Description values reflect the OGIS specification.

Table 2–1 Valid SDO_GTYPE Values

Value	Geometry Type	Description
<i>dl00</i>	UNKNOWN_GEOMETRY	Spatial ignores this geometry.
<i>dl01</i>	POINT	Geometry contains one point.
<i>dl02</i>	LINE or CURVE	Geometry contains one line string that can contain straight or circular arc segments, or both. (LINE and CURVE are synonymous in this context.)
<i>dl03</i>	POLYGON	Geometry contains one polygon with or without holes. ¹
<i>dl04</i>	COLLECTION	Geometry is a heterogeneous collection of elements. COLLECTION is a superset that includes all other types.
<i>dl05</i>	MULTIPOINT	Geometry has one or more points. (MULTIPOINT is a superset of POINT.)
<i>dl06</i>	MULTILINE or MULTICURVE	Geometry has one or more line strings. (MULTILINE and MULTICURVE are synonymous in this context, and each is a superset of both LINE and CURVE.)
<i>dl07</i>	MULTIPOLYGON	Geometry can have multiple, disjoint polygons (more than one exterior boundary). (MULTIPOLYGON is a superset of POLYGON.)

¹ For a polygon with holes, enter the exterior boundary first, followed by any interior boundaries.

The *d* in the Value column of [Table 2–1](#) is the number of dimensions: 2, 3, or 4. For example, an SDO_GTYPE value of 2003 indicates a two-dimensional polygon.

Note: The 1-digit SDO_GTYPE values from before release 8.1.6 value are still supported. If a 1-digit value is used, however, Oracle Spatial determines the number of dimensions from the DIMINFO column of the metadata views, described in [Section 2.6.3](#).

Also, if 1-digit SDO_GTYPE values are converted to 4-digit values, any SDO_GTYPE values that end in 3 or 5 in the SDO_ELEM_INFO array (described in [Section 2.2.4](#)) must also be converted.

The number of dimensions reflects the number of ordinates used to represent each vertex (for example, X,Y for two-dimensional objects). Points and lines are considered two-dimensional objects. (However, see [Section 7.2](#) for dimension information about LRS points.)

In any given layer (column), all geometries must have the same number of dimensions. For example, you cannot mix two-dimensional and three-dimensional data in the same layer.

The following methods are available for returning the individual *dltt* components of the SDO_GTYPE for a geometry object: `Get_Dims`, `Get_LRS_Dim`, and `Get_Gtype`. These methods are described in [Section 2.3](#).

2.2.2 SDO_SRID

The SDO_SRID attribute can be used to identify a coordinate system (spatial reference system) to be associated with the geometry. If SDO_SRID is null, no coordinate system is associated with the geometry. If SDO_SRID is not null, it must contain a value from the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)), and this value must be inserted into the SRID column of the USER_SDO_GEOM_METADATA view (described in [Section 2.6](#)).

All geometries in a geometry column must have the same SDO_SRID value.

For information about coordinate systems, see [Chapter 6](#).

2.2.3 SDO_POINT

The SDO_POINT attribute is defined using the SDO_POINT_TYPE object type, which has the attributes X, Y, and Z, all of type NUMBER. (The SDO_POINT_TYPE definition is shown in [Section 2.2](#).) If the SDO_ELEM_INFO and SDO_ORDINATES arrays are both null, and the SDO_POINT attribute is non-null, then the X and Y values are considered to be the coordinates for a point geometry. Otherwise, the SDO_POINT attribute is ignored by Spatial. You should store point geometries in the SDO_POINT attribute for optimal storage; and if you have only point geometries in a layer, it is strongly recommended that you store the point geometries in the SDO_POINT attribute.

[Section 2.5.5](#) illustrates a point geometry and provides examples of inserting and querying point geometries.

Note: Do not use the SDO_POINT attribute in defining a linear referencing system (LRS) point or an oriented point. For information about LRS, see [Chapter 7](#). For information about oriented points, see [Section 2.5.6](#).

2.2.4 SDO_ELEM_INFO

The SDO_ELEM_INFO attribute is defined using a varying length array of numbers. This attribute lets you know how to interpret the ordinates stored in the SDO_ORDINATES attribute (described in [Section 2.2.5](#)).

Each triplet set of numbers is interpreted as follows:

- SDO_STARTING_OFFSET -- Indicates the offset within the SDO_ORDINATES array where the first ordinate for this element is stored. Offset values start at 1 and not at 0. Thus, the first ordinate for the first element will be at SDO_GEOMETRY.SDO_ORDINATES(1). If there is a second element, its first ordinate will be at SDO_GEOMETRY.SDO_ORDINATES(*n*), where *n* reflects the position within the SDO_ORDINATE_ARRAY definition (for example, 19 for the 19th number, as in [Figure 2-3](#) in [Section 2.5.2](#)).

- SDO_ETYPE -- Indicates the type of the element. Valid values are shown in [Table 2–2](#).

SDO_ETYPE values 1, 2, 1003, and 2003 are considered *simple elements*. They are defined by a single triplet entry in the SDO_ELEM_INFO array. For SDO_ETYPE values 1003 and 2003, the first digit indicates *exterior* (1) or *interior* (2):

1003: exterior polygon ring (must be specified in counterclockwise order)

2003: interior polygon ring (must be specified in clockwise order)

Note: The use of 3 as an SDO_ETYPE value for polygon ring elements in a single geometry is discouraged. You should specify 3 only if you do not know if the simple polygon is exterior or interior, and you should then upgrade the table or layer to the current format using the [SDO_MIGRATE.TO_CURRENT](#) procedure, described in [Chapter 17](#).

You cannot mix 1-digit and 4-digit SDO_ETYPE values in a single geometry. If you use 4-digit SDO_ETYPE values, you must use 4-digit SDO_GTYPE values.

SDO_ETYPE values 4, 1005, and 2005 are considered *compound elements*. They contain at least one header triplet with a series of triplet values that belong to the compound element. For SDO_ETYPE values 1005 and 2005, the first digit indicates *exterior* (1) or *interior* (2):

1005: exterior polygon ring (must be specified in counterclockwise order)

2005: interior polygon ring (must be specified in clockwise order)

Note: The use of 5 as an SDO_ETYPE value for polygon ring elements in a single geometry is discouraged. You should specify 5 only if you do not know if the compound polygon is exterior or interior, and you should then upgrade the table or layer to the current format using the [SDO_MIGRATE.TO_CURRENT](#) procedure, described in [Chapter 17](#).

You cannot mix 1-digit and 4-digit SDO_ETYPE values in a single geometry. If you use 4-digit SDO_ETYPE values, you must use 4-digit SDO_GTYPE values.

The elements of a compound element are contiguous. The last point of a subelement in a compound element is the first point of the next subelement. The point is not repeated.

- SDO_INTERPRETATION -- Means one of two things, depending on whether or not SDO_ETYPE is a compound element.

If SDO_ETYPE is a compound element (4, 1005, or 2005), this field specifies how many subsequent triplet values are part of the element.

If the SDO_ETYPE is not a compound element (1, 2, 1003, or 2003), the interpretation attribute determines how the sequence of ordinates for this element is interpreted. For example, a line string or polygon boundary may be made up of a sequence of connected straight line segments or circular arcs.

Descriptions of valid SDO_ETYPE and SDO_INTERPRETATION value pairs are given in [Table 2-2](#).

If a geometry consists of more than one element, then the last ordinate for an element is always one less than the starting offset for the next element. The last element in the geometry is described by the ordinates from its starting offset to the end of the SDO_ORDINATES varying length array.

For compound elements (SDO_ETYPE values 4, 1005, or 2005), a set of n triplets (one for each subelement) is used to describe the element. It is important to remember that subelements of a compound element are contiguous. The last point of a subelement is the first point of the next subelement. For subelements 1 through $n-1$, the end point of one subelement is the same as the starting point of the next subelement. The starting point for subelements 2... $n-2$ is the same as the end point of subelement 1... $n-1$. The last ordinate of subelement n is either the starting offset minus 1 of the next element in the geometry, or the last ordinate in the SDO_ORDINATES varying length array.

The current size of a varying length array can be determined by using the function `varray_variable.Count` in PL/SQL or `OCICollSize` in the Oracle Call Interface (OCI).

The semantics of each SDO_ETYPE element and the relationship between the SDO_ELEM_INFO and SDO_ORDINATES varying length arrays for each of these SDO_ETYPE elements are given in [Table 2-2](#).

Table 2-2 Values and Semantics in SDO_ELEM_INFO

SDO_ETYPE	SDO_INTERPRETATION	Meaning
0	(any numeric value)	Type 0 (zero) element. Used to model geometry types not supported by Oracle Spatial. For more information, see Section 2.5.7 .
1	1	Point type.
1	0	Orientation for an oriented point. For more information, see Section 2.5.6 .
1	$n > 1$	Point cluster with n points.
2	1	Line string whose vertices are connected by straight line segments.
2	2	Line string made up of a connected sequence of circular arcs. Each circular arc is described using three coordinates: the start point of the arc, any point on the arc, and the end point of the arc. The coordinates for a point designating the end of one arc and the start of the next arc are not repeated. For example, five coordinates are used to describe a line string made up of two connected circular arcs. Points 1, 2, and 3 define the first arc, and points 3, 4, and 5 define the second arc, where point 3 is only stored once.
1003 or 2003	1	Simple polygon whose vertices are connected by straight line segments. You must specify a point for each vertex, and the last point specified must be exactly the same point as the first (to close the polygon), regardless of the tolerance value. For example, for a 4-sided polygon, specify 5 points, with point 5 the same as point 1.

Table 2–2 (Cont.) Values and Semantics in SDO_ELEM_INFO

SDO_ETYPE	SDO_INTERPRETATION	Meaning
1003 or 2003	2	<p>Polygon made up of a connected sequence of circular arcs that closes on itself. The end point of the last arc is the same as the start point of the first arc.</p> <p>Each circular arc is described using three coordinates: the start point of the arc, any point on the arc, and the end point of the arc. The coordinates for a point designating the end of one arc and the start of the next arc are not repeated. For example, five coordinates are used to describe a polygon made up of two connected circular arcs. Points 1, 2, and 3 define the first arc, and points 3, 4, and 5 define the second arc. The coordinates for points 1 and 5 must be the same (tolerance is not considered), and point 3 is not repeated.</p>
1003 or 2003	3	<p>Rectangle type (sometimes called <i>optimized rectangle</i>). A bounding rectangle such that only two points, the lower-left and the upper-right, are required to describe it. The rectangle type can be used with geodetic or non-geodetic data. However, with geodetic data, use this type only to create a query window (not for storing objects in the database). For detailed information about using this type with geodetic data, including examples, see Section 6.2.3.</p>
1003 or 2003	4	<p>Circle type. Described by three distinct non-colinear points, all on the circumference of the circle.</p>
4	$n > 1$	<p>Compound line string with some vertices connected by straight line segments and some by circular arcs. The value n in the Interpretation column specifies the number of contiguous subelements that make up the line string.</p> <p>The next n triplets in the SDO_ELEM_INFO array describe each of these subelements. The subelements can only be of SDO_ETYPE 2. The last point of a subelement is the first point of the next subelement, and must not be repeated.</p> <p>See Section 2.5.3 and Figure 2–4 for an example of a compound line string geometry.</p>
1005 or 2005	$n > 1$	<p>Compound polygon with some vertices connected by straight line segments and some by circular arcs. The value n in the Interpretation column specifies the number of contiguous subelements that make up the polygon.</p> <p>The next n triplets in the SDO_ELEM_INFO array describe each of these subelements. The subelements can only be of SDO_ETYPE 2. The end point of a subelement is the start point of the next subelement, and it must not be repeated. The start and end points of the polygon must be exactly the same point (tolerance is ignored).</p> <p>See Section 2.5.4 and Figure 2–5 for an example of a compound polygon geometry.</p>

2.2.5 SDO_ORDINATES

The SDO_ORDINATES attribute is defined using a varying length array (1048576) of NUMBER type that stores the coordinate values that make up the boundary of a spatial object. This array must always be used in conjunction with the SDO_ELEM_INFO varying length array. The values in the array are ordered by dimension. For example, a polygon whose boundary has four two-dimensional points is stored as {X1, Y1, X2, Y2, X3, Y3, X4, Y4, X1, Y1}. If the points are three-dimensional, then they are stored as {X1, Y1, Z1, X2, Y2, Z2, X3, Y3, Z3, X4, Y4, Z4, X1, Y1, Z1}. Spatial index

creation, operators, and functions ignore the Z values because this release of the product supports only two-dimensional spatial objects. The number of dimensions associated with each point is stored as metadata in the xxx_SDO_GEOM_METADATA views, described in [Section 2.6](#).

The values in the SDO_ORDINATES array must all be valid and non-null. There are no special values used to delimit elements in a multielement geometry. The start and end points for the sequence describing a specific element are determined by the STARTING_OFFSET values for that element and the next element in the SDO_ELEM_INFO array, as explained in [Section 2.2.4](#). The offset values start at 1. SDO_ORDINATES(1) is the first ordinate of the first point of the first element.

2.2.6 Usage Considerations

You should use the SDO_GTYPE values as shown in [Table 2–1](#); however, Spatial does not check or enforce all geometry consistency constraints. Spatial does check the following:

- For SDO_GTYPE values *d001* and *d005*, any subelement not of SDO_ETYPE 1 is ignored.
- For SDO_GTYPE values *d002* and *d006*, any subelement not of SDO_ETYPE 2 or 4 is ignored.
- For SDO_GTYPE values *d003* and *d007*, any subelement not of SDO_ETYPE 3 or 5 is ignored. (This includes SDO_ETYPE variants 1003, 2003, 1005, and 2005, which are explained in [Section 2.2.4](#)).

The [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#) function can be used to evaluate the consistency of a single geometry object or of all geometry objects in a specified feature table.

2.3 SDO_GEOMETRY Methods

The SDO_GEOMETRY object type (described in [Section 2.2](#)) has methods (member functions) that retrieve information about a geometry object. [Table 2–3](#) lists these methods.

Table 2–3 SDO_GEOMETRY Methods

Name	Returns	Description
Get_Dims	NUMBER	Returns the number of dimensions of a geometry object, as specified in its SDO_GTYPE value. In Oracle Spatial, the Get_Dims and ST_CoordDim methods return the same result.
Get_GType	NUMBER	Returns the geometry type of a geometry object, as specified in its SDO_GTYPE value.
Get_LRS_Dim	NUMBER	Returns the measure dimension of an LRS geometry object, as specified in its SDO_GTYPE value. A return value of 0 indicates that the geometry is a standard (non-LRS) geometry, or is an LRS geometry in the format before release 9.0.1 and with measure as the default (last) dimension; 3 indicates that the third dimension contains the measure information; 4 indicates that the fourth dimension contains the measure information.
Get_WKB	BLOB	Returns the well-known binary (WKB) format of a geometry object. (The returned object does not include any SRID information.)

Table 2–3 (Cont.) SDO_GEOMETRY Methods

Name	Returns	Description
Get_WKT	CLOB	Returns the well-known text (WKT) format (explained in Section 6.7.1.1) of a geometry object. (The returned object does not include any SRID information.)
ST_CoordDim	NUMBER	Returns the coordinate dimension (as defined by the ISO/IEC SQL Multimedia standard) of a geometry object. In Oracle Spatial, the Get_Dims and ST_CoordDim methods return the same result.
ST_IsValid	NUMBER	Returns 0 if a geometry object is invalid or 1 if it is valid. (The ISO/IEC SQL Multimedia standard uses the term <i>well formed</i> for <i>valid</i> in this context.) This method uses 0.001 as the tolerance value. (Tolerance is explained in Section 1.5.5 .) To specify a different tolerance value or to learn more about why a geometry is invalid, use the SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT function, which is documented in Chapter 15 .

The geometry must have a 4-digit SDO_GTYPE value in the format *dltt*, as described in [Section 2.2.1](#).

[Example 2–2](#) shows most of the SDO_GEOMETRY methods. (The Get_WKB method is not included because its output cannot be displayed by SQL*Plus.)

Example 2–2 SDO_GEOMETRY Methods

```
SELECT c.shape.Get_Dims()
   FROM cola_markets c WHERE c.name = 'cola_b';

C.SHAPE.GET_DIMS()
-----
                2

SELECT c.shape.Get_GType()
   FROM cola_markets c WHERE c.name = 'cola_b';

C.SHAPE.GET_GTYPE()
-----
                3

SELECT a.route_geometry.Get_LRS_Dim()
   FROM lrs_routes a WHERE a.route_id = 1;

A.ROUTE_GEOMETRY.GET_LRS_DIM()
-----
                3

SELECT c.shape.Get_WKT()
   FROM cola_markets c WHERE c.name = 'cola_b';

C.SHAPE.GET_WKT()
-----
POLYGON ((5.0 1.0, 8.0 1.0, 8.0 6.0, 5.0 7.0, 5.0 1.0))

SELECT c.shape.ST_CoordDim()
   FROM cola_markets c WHERE c.name = 'cola_b';

C.SHAPE.ST_COORDDIM()
-----
```

```

SELECT c.shape.ST_IsValid()
       FROM cola_markets c WHERE c.name = 'cola_b';

C.SHAPE.ST_ISVALID()
-----
1

```

2.4 SDO_GEOMETRY Constructors

The SDO_GEOMETRY object type (described in [Section 2.2](#)) has constructors that create a geometry object from a well-known text (WKT) string in CLOB or VARCHAR2 format, or from a well-known binary (WKB) object in BLOB format. The following constructor formats are available:

```

SDO_GEOMETRY(wkt CLOB, srid NUMBER DEFAULT NULL);
SDO_GEOMETRY(wkt VARCHAR2, srid NUMBER DEFAULT NULL);
SDO_GEOMETRY(wkb BLOB, srid NUMBER DEFAULT NULL);

```

If the created geometry is inserted into a table, the SRID value used with the constructor must match the SDO_SRID value of the geometries in the table.

The following simple example constructs a point geometry using a well-known text string. (In a WKT, spaces separate ordinates of a vertex, and commas separate vertices.)

```

SELECT SDO_GEOMETRY('POINT(-79 37)') FROM DUAL;

SDO_GEOMETRY('POINT(-7937)')(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_I
-----
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(-79, 37, NULL), NULL, NULL)

```

[Example 2-3](#) shows SDO_GEOMETRY constructors that create geometry objects, insert the objects into a table, and display the objects that were added to the table.

Example 2-3 SDO_GEOMETRY Constructors to Create Geometries

```

DECLARE
  cola_b_wkb  BLOB;
  cola_b_wkt_clob  CLOB;
  cola_b_wkt_varchar  VARCHAR2(255);
  cola_b_geom  SDO_GEOMETRY;

BEGIN
  -- Get cola_b geometry into CLOB, VARCHAR2, and BLOB objects,
  -- for use by the constructor.
  SELECT c.shape.Get_WKT() INTO cola_b_wkt_clob
     FROM cola_markets c WHERE c.name = 'cola_b';
  cola_b_wkt_varchar := cola_b_wkt_clob;
  SELECT c.shape.Get_WKB() INTO cola_b_wkb
     FROM cola_markets c WHERE c.name = 'cola_b';

  -- Use some SDO_GEOMETRY constructors;
  -- insert 3 geometries into the table; display the geometries later.
  cola_b_geom := SDO_GEOMETRY(cola_b_wkt_clob);
  INSERT INTO cola_markets VALUES (101, 'cola_b_from_clob', cola_b_geom);
  cola_b_geom := SDO_GEOMETRY(cola_b_wkt_varchar);
  INSERT INTO cola_markets VALUES (102, 'cola_b_from_varchar', cola_b_geom);
  cola_b_geom := SDO_GEOMETRY(cola_b_wkb);

```

```

INSERT INTO cola_markets VALUES (103, 'cola_b_from_wkb', cola_b_geom);
END;
/

PL/SQL procedure successfully completed.

-- Display the geometries created using SDO_GEOMETRY constructors.
-- All three geometries are identical.
SELECT name, shape FROM cola_markets WHERE mkt_id > 100;

NAME
-----
SHAPE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
cola_b_from_clob
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(5, 1, 8, 1, 8, 6, 5, 7, 5, 1))

cola_b_from_varchar
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(5, 1, 8, 1, 8, 6, 5, 7, 5, 1))

cola_b_from_wkb
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(5, 1, 8, 1, 8, 6, 5, 7, 5, 1))

```

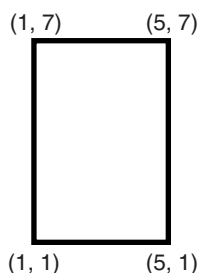
2.5 Geometry Examples

This section contains examples of many geometry types:

- [Section 2.5.1, "Rectangle"](#)
- [Section 2.5.2, "Polygon with a Hole"](#)
- [Section 2.5.3, "Compound Line String"](#)
- [Section 2.5.4, "Compound Polygon"](#)
- [Section 2.5.5, "Point"](#)
- [Section 2.5.6, "Oriented Point"](#)
- [Section 2.5.7, "Type 0 \(Zero\) Element"](#)
- [Section 2.5.8, "Several Geometry Types"](#)

2.5.1 Rectangle

[Figure 2–2](#) illustrates the rectangle that represents `cola_a` in the example in [Section 2.1](#).

Figure 2–2 Rectangle

In the SDO_GEOMETRY definition of the geometry illustrated in [Figure 2–2](#):

- SDO_GTYPE = 2003. The 2 indicates two-dimensional, and the 3 indicates a polygon.
- SDO_SRID = NULL.
- SDO_POINT = NULL.
- SDO_ELEM_INFO = (1, 1003, 3). The final 3 in 1,1003,3 indicates that this is a rectangle. Because it is a rectangle, only two ordinates are specified in SDO_ORDINATES (lower-left and upper-right).
- SDO_ORDINATES = (1,1, 5,7). These identify the lower-left and upper-right ordinates of the rectangle.

[Example 2–4](#) shows a SQL statement that inserts the geometry illustrated in [Figure 2–2](#) into the database.

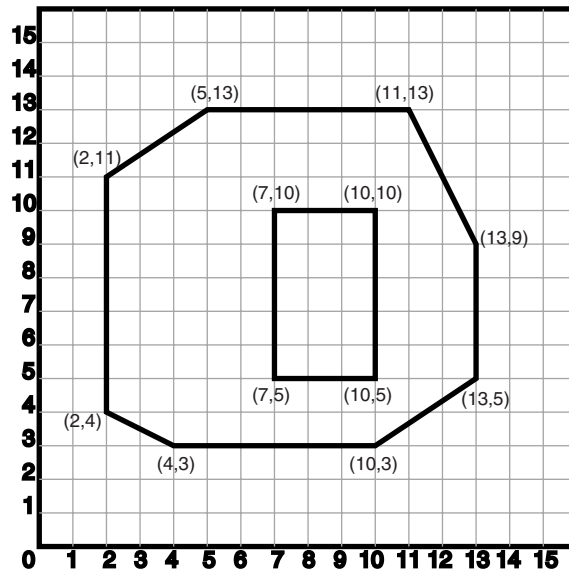
Example 2–4 SQL Statement to Insert a Rectangle

```
INSERT INTO cola_markets VALUES (
  1,
  'cola_a',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
    SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
      -- define rectangle (lower left and upper right) with
      -- Cartesian-coordinate data
  )
);
```

2.5.2 Polygon with a Hole

[Figure 2–3](#) illustrates a polygon consisting of two elements: an exterior polygon ring and an interior polygon ring. The inner element in this example is treated as a void (a hole).

Figure 2–3 Polygon with a Hole



In the SDO_GEOMETRY definition of the geometry illustrated in [Figure 2–3](#):

- SDO_GTYPE = 2003. The 2 indicates two-dimensional, and the 3 indicates a polygon.
- SDO_SRID = NULL.
- SDO_POINT = NULL.
- SDO_ELEM_INFO = (1,1003,1, 19,2003,1). There are two triplet elements: 1,1003,1 and 19,2003,1.

1003 indicates that the element is an exterior polygon ring; 2003 indicates that the element is an interior polygon ring.

19 indicates that the second element (the interior polygon ring) ordinate specification starts at the 19th number in the SDO_ORDINATES array (that is, 7, meaning that the first point is 7,5).

- SDO_ORDINATES = (2,4, 4,3, 10,3, 13,5, 13,9, 11,13, 5,13, 2,11, 2,4, 7,5, 7,10, 10,10, 10,5, 7,5).
- The area ([SDO_GEOM.SDO_AREA](#) function) of the polygon is the area of the exterior polygon minus the area of the interior polygon. In this example, the area is 84 (99 - 15).
- The perimeter ([SDO_GEOM.SDO_LENGTH](#) function) of the polygon is the perimeter of the exterior polygon plus the perimeter of the interior polygon. In this example, the perimeter is 52.9193065 (36.9193065 + 16).

[Example 2–5](#) shows a SQL statement that inserts the geometry illustrated in [Figure 2–3](#) into the database.

Example 2–5 SQL Statement to Insert a Polygon with a Hole

```
INSERT INTO cola_markets VALUES(
  10,
  'polygon_with_hole',
  SDO_GEOMETRY(
```

```

2003, -- two-dimensional polygon
NULL,
NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1, 19,2003,1), -- polygon with hole
SDO_ORDINATE_ARRAY(2,4, 4,3, 10,3, 13,5, 13,9, 11,13, 5,13, 2,11, 2,4,
                    7,5, 7,10, 10,10, 10,5, 7,5)
)
);

```

An example of such a "polygon with a hole" might be a land mass (such as a country or an island) with a lake inside it. Of course, an actual land mass might have many such interior polygons: each one would require a triplet element in SDO_ELEM_INFO, plus the necessary ordinate specification.

Exterior and interior rings cannot be nested. For example, if a country has a lake and there is an island in the lake (and perhaps a lake on the island), a separate polygon must be defined for the island; the island cannot be defined as an interior polygon ring within the interior polygon ring of the lake.

In a **multipolygon** (polygon collection), rings must be grouped by polygon, and the first ring of each polygon must be the exterior ring. For example, consider a polygon collection that contains two polygons (A and B):

- Polygon A (one interior "hole"): exterior ring A0, interior ring A1
- Polygon B (two interior "holes"): exterior ring B0, interior ring B1, interior ring B2

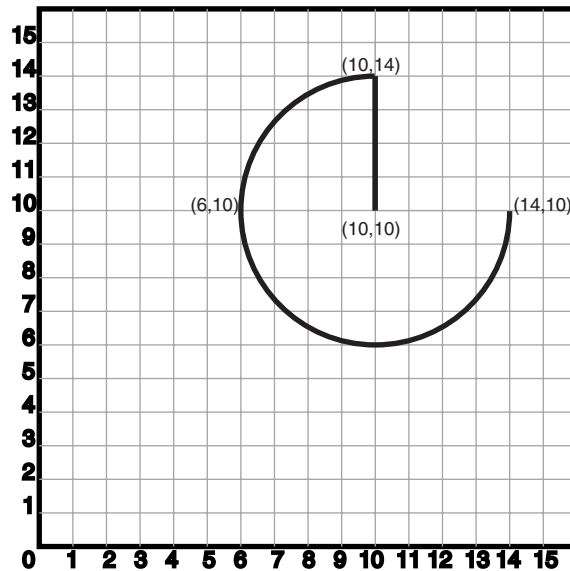
The elements in SDO_ELEM_INFO and SDO_ORDINATES must be in one of the following orders (depending on whether you specify Polygon A or Polygon B first):

- A0, A1; B0, B1, B2
- B0, B1, B2; A0, A1

2.5.3 Compound Line String

Figure 2–4 illustrates a crescent-shaped object represented as a compound line string made up of one straight line segment and one circular arc. Four points are required to represent this shape: points (10,10) and (10,14) describe the straight line segment, and points (10,14), (6,10), and (14,10) describe the circular arc.

Figure 2–4 Compound Line String



In the SDO_GEOMETRY definition of the geometry illustrated in [Figure 2–4](#):

- SDO_GTYPE = 2002. The first 2 indicates two-dimensional, and the second 2 indicates one or more line segments.
- SDO_SRID = NULL.
- SDO_POINT = NULL.
- SDO_ELEM_INFO = (1,4,2, 1,2,1, 3,2,2). There are three triplet elements: 1,4,2, 1,2,1, and 3,2,2.

The first triplet indicates that this element is a compound line string made up of two subelement line strings, which are described with the next two triplets.

The second triplet indicates that the line string is made up of straight line segments and that the ordinates for this line string start at offset 1. The end point of this line string is determined by the starting offset of the second line string, 3 in this instance.

The third triplet indicates that the second line string is made up of circular arcs with ordinates starting at offset 3. The end point of this line string is determined by the starting offset of the next element or the current length of the SDO_ORDINATES array, if this is the last element.

- SDO_ORDINATES = (10,10, 10,14, 6,10, 14,10).

[Example 2–6](#) shows a SQL statement that inserts the geometry illustrated in [Figure 2–4](#) into the database.

Example 2–6 SQL Statement to Insert a Compound Line String

```
INSERT INTO cola_markets VALUES(
  11,
  'compound_line_string',
  SDO_GEOMETRY(
    2002,
    NULL,
    NULL,
```



```

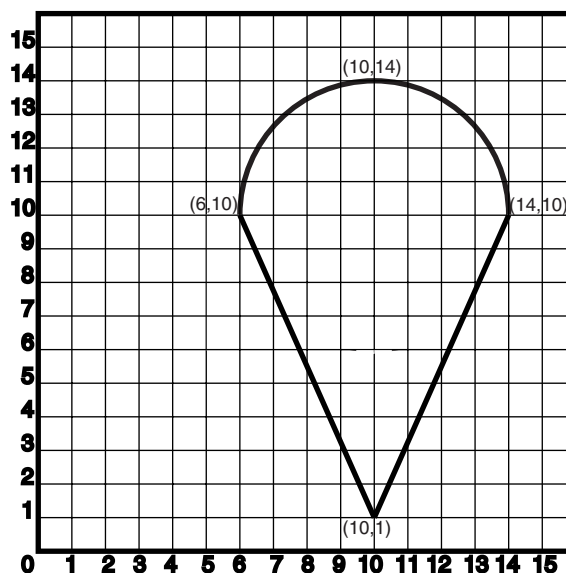
SDO_ELEM_INFO_ARRAY(1,4,2, 1,2,1, 3,2,2), -- compound line string
SDO_ORDINATE_ARRAY(10,10, 10,14, 6,10, 14,10)
)
);

```

2.5.4 Compound Polygon

Figure 2-5 illustrates an ice cream cone-shaped object represented as a compound polygon made up of one straight line segment and one circular arc. Five points are required to represent this shape: points (6,10), (10,1), and (14,10) describe one acute angle-shaped line string, and points (14,10), (10,14), and (6,10) describe the circular arc. The starting point of the line string and the ending point of the circular arc are the same point (6,10). The SDO_ELEM_INFO array contains three triplets for this compound line string. These triplets are {(1,1005,2), (1,2,1), (5,2,2)}.

Figure 2-5 Compound Polygon



In the SDO_GEOMETRY definition of the geometry illustrated in Figure 2-5:

- SDO_GTYPE = 2003. The 2 indicates two-dimensional, and the 3 indicates a polygon.
- SDO_SRID = NULL.
- SDO_POINT = NULL.
- SDO_ELEM_INFO = (1,1005,2, 1,2,1, 5,2,2). There are three triplet elements: 1,1005,2, 1,2,1, and 5,2,2.

The first triplet indicates that this element is a compound polygon made up of two subelement line strings, which are described using the next two triplets.

The second triplet indicates that the first subelement line string is made up of straight line segments and that the ordinates for this line string start at offset 1. The end point of this line string is determined by the starting offset of the second line string, 5 in this instance. Because the vertices are two-dimensional, the coordinates for the end point of the first line string are at ordinates 5 and 6.

The third triplet indicates that the second subelement line string is made up of a circular arc with ordinates starting at offset 5. The end point of this line string is determined by the starting offset of the next element or the current length of the SDO_ORDINATES array, if this is the last element.

- SDO_ORDINATES = (6,10, 10,1, 14,10, 10,14, 6,10).

Example 2-7 shows a SQL statement that inserts the geometry illustrated in Figure 2-5 into the database.

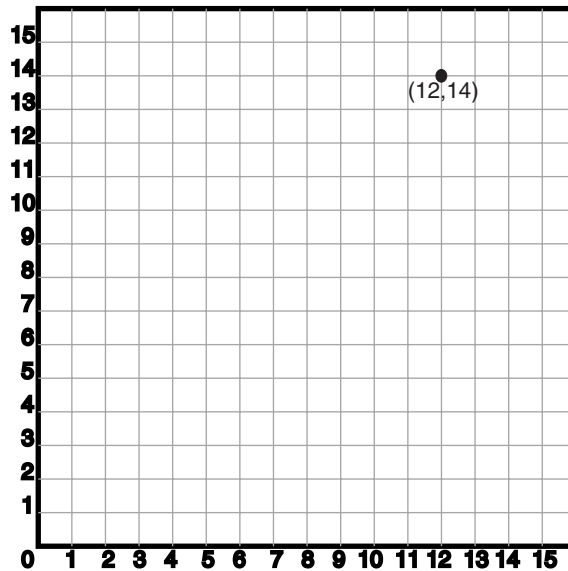
Example 2-7 SQL Statement to Insert a Compound Polygon

```
INSERT INTO cola_markets VALUES(
  12,
  'compound_polygon',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1005,2, 1,2,1, 5,2,2), -- compound polygon
    SDO_ORDINATE_ARRAY(6,10, 10,1, 14,10, 10,14, 6,10)
  )
);
```

2.5.5 Point

Figure 2-6 illustrates a point-only geometry at coordinates (12,14).

Figure 2-6 Point-Only Geometry



In the SDO_GEOMETRY definition of the geometry illustrated in Figure 2-6:

- SDO_GTYPE = 2001. The 2 indicates two-dimensional, and the 1 indicates a single point.
- SDO_SRID = NULL.

- `SDO_POINT = SDO_POINT_TYPE(12, 14, NULL)`. The `SDO_POINT` attribute is defined using the `SDO_POINT_TYPE` object type, because this is a point-only geometry.

For more information about the `SDO_POINT` attribute, see [Section 2.2.3](#).

- `SDO_ELEM_INFO` and `SDO_ORDINATES` are both `NULL`, as required if the `SDO_POINT` attribute is specified.

[Example 2-8](#) shows a SQL statement that inserts the geometry illustrated in [Figure 2-6](#) into the database.

Example 2-8 SQL Statement to Insert a Point-Only Geometry

```
INSERT INTO cola_markets VALUES(
  90,
  'point_only',
  SDO_GEOMETRY(
    2001,
    NULL,
    SDO_POINT_TYPE(12, 14, NULL),
    NULL,
    NULL));
```

You can search for point-only geometries based on the X, Y, and Z values in the `SDO_POINT_TYPE` specification. [Example 2-9](#) is a query that asks for all points whose first coordinate (the X value) is 12, and it finds the point that was inserted in [Example 2-8](#).

Example 2-9 Query for Point-Only Geometry Based on a Coordinate Value

```
SELECT * from cola_markets c WHERE c.shape.SDO_POINT.X = 12;

MKT_ID NAME
-----
SHAPE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
          90 point_only
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(12, 14, NULL), NULL, NULL)
```

2.5.6 Oriented Point

An **oriented point** is a special type of point geometry that includes coordinates representing the locations of the point and a virtual end point, to indicate an orientation vector that can be used for rotating a symbol at the point or extending a label from the point. The main use for an oriented point is in map visualization and display applications that include symbols, such as a shield symbol to indicate a highway.

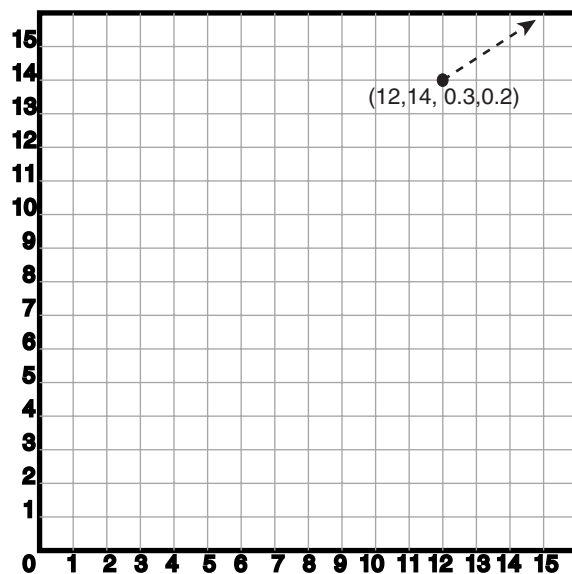
To specify an oriented point:

- Use an `SDO_GTYPE` value (explained in [Section 2.2.1](#)) for a point or multipoint geometry.
- Specify a null value for the `SDO_POINT` attribute.
- In the `SDO_ELEM_INFO` array (explained in [Section 2.2.4](#)), specify an additional triplet, with the second and third values (`SDO_ETYPE` and `SDO_INTERPRETATION`) as 1 and 0. For example, a triplet of 3,1,0 indicates that the point is an oriented point, with the third number in the `SDO_ORDINATES` array being the first coordinate, or x-axis value, of the end point reflecting the orientation vector for any symbol or label.

- In the SDO_ORDINATES array (explained in [Section 2.2.5](#)), specify the coordinates of the end point for the orientation vector from the point, with values between -1 and 1. The orientation start point is assumed to be (0,0), and it is translated to the location of the physical point to which it corresponds.

Figure 2–7 illustrates an oriented point geometry at coordinates (12,14), with an orientation vector of approximately 34 degrees (counterclockwise from the x-axis), reflecting the orientation coordinates 0.3,0.2. (To have an orientation that more precisely matches a specific angle, refer to the cotangent or tangent values in the tables in a trigonometry textbook.) The orientation vector in this example goes from (0,0) to (0.3,0.2) and extends onward. Assuming $i=0.3$ and $j=0.2$, the angle in radians can be calculated as follows: angle in radians = $\arctan(j/i)$. The angle is then applied to the physical point associated with the orientation vector.

Figure 2–7 Oriented Point Geometry



In the SDO_GEOMETRY definition of the geometry illustrated in [Figure 2–7](#):

- SDO_GTYPE = 2001. The 2 indicates two-dimensional, and the 1 indicates a single point.
- SDO_SRID = NULL.
- SDO_POINT = NULL.
- SDO_ELEM_INFO = (1,1,1, 3,1,0). The final 1,0 in 3,1,0 indicates that this is an oriented point.
- SDO_ORDINATES = (12,14, 0.3,0.2). The 12,14 identifies the physical coordinates of the point; and the 0.3,0.2 identifies the x and y coordinates (assuming 12,14 as the origin) of the end point of the orientation vector. The resulting orientation vector slopes upward at about a 34-degree angle.

[Example 2–10](#) shows a SQL statement that inserts the geometry illustrated in [Figure 2–7](#) into the database.

Example 2–10 SQL Statement to Insert an Oriented Point Geometry

```
INSERT INTO cola_markets VALUES (
    91,
```

```
'oriented_point',
SDO_GEOMETRY(
  2001,
  NULL,
  NULL,
  SDO_ELEM_INFO_ARRAY(1,1,1, 3,1,0),
  SDO_ORDINATE_ARRAY(12,14, 0.3,0.2));
```

The following guidelines apply to the definition of an oriented point:

- The numbers defining the orientation vector must be between -1 and 1. (In [Example 2–10](#), these numbers are 0.3 and 0.2.)
- Multipoint oriented points are allowed (see [Example 2–11](#)), but the orientation information must follow the point being oriented.

The following considerations apply to the dimensionality of the orientation vector for an oriented point:

- A two-dimensional point has a two-dimensional orientation vector.
- A two-dimensional point with an LRS measure (SDO_GTYPE=3301) has a two-dimensional orientation vector.
- A three-dimensional point (SDO_GTYPE=3001) has a three-dimensional orientation vector.
- A three-dimensional point with an LRS measure (SDO_GTYPE=4401) has a three-dimensional orientation vector.
- A four-dimensional point (SDO_GTYPE=4001) has a three-dimensional orientation vector.

[Example 2–11](#) shows a SQL statement that inserts an oriented multipoint geometry into the database. The multipoint geometry contains two points, at coordinates (12,14) and (12, 10), with the two points having different orientation vectors. The statement is similar to the one in [Example 2–10](#), but in [Example 2–11](#) the second point has an orientation vector pointing down and to the left at 45 degrees (or, 135 degrees clockwise from the x-axis), reflecting the orientation coordinates -1,-1.

Example 2–11 SQL Statement to Insert an Oriented Multipoint Geometry

```
-- Oriented multipoint: 2 points, different orientations
INSERT INTO cola_markets VALUES(
  92,
  'oriented_multipoint',
  SDO_GEOMETRY(
    2005, -- Multipoint
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1,1, 3,1,0, 5,1,1, 7,1,0),
    SDO_ORDINATE_ARRAY(12,14, 0.3,0.2, 12,10, -1,-1));
```

2.5.7 Type 0 (Zero) Element

Type 0 (zero) elements are used to model geometry types that are not supported by Oracle Spatial, such as curves and splines. A type 0 element has an SDO_ETYPE value of 0. (See [Section 2.2.4](#) for information about the SDO_ETYPE.) Type 0 elements are not indexed by Oracle Spatial, and they are ignored by Spatial functions and procedures.

Geometries with type 0 elements must contain at least one nonzero element, that is, an element with an SDO_ETYPE value that is not 0. The nonzero element should be an approximation of the unsupported geometry, and therefore it must have both:

- An SDO_ETYPE value associated with a geometry type supported by Spatial
- An SDO_INTERPRETATION value that is valid for the SDO_ETYPE value (see [Table 2-2](#))

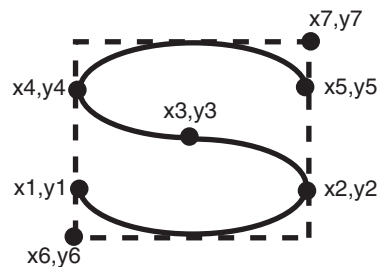
(The SDO_INTERPRETATION value for the type 0 element can be any numeric value, and applications are responsible for determining the validity and significance of the value.)

The nonzero element is indexed by Spatial, and it will be returned by the spatial index.

The SDO_GTYPE value for a geometry containing a type 0 element must be set to the value for the geometry type of the nonzero element.

[Figure 2-8](#) shows a geometry with two elements: a curve (unsupported geometry) and a rectangle (the nonzero element) that approximates the curve. The curve looks like the letter S, and the rectangle is represented by the dashed line.

Figure 2-8 Geometry with Type 0 (Zero) Element



In the example shown in [Figure 2-8](#):

- The SDO_GTYPE value for the geometry is 2003 (for a two-dimensional polygon).
- The SDO_ELEM_INFO array contains two triplets for this compound line string. For example, the triplets might be {(1,0,57), (11,1003,3)}. That is:

Ordinate Starting Offset (SDO_STARTING_OFFSET)	Element Type (SDO_ETYPE)	Interpretation (SDO_INTERPRETATION)
1	0	57
11	1003	3

In this example:

- The type 0 element has an SDO_ETYPE value of 0.
- The nonzero element (rectangle) has an SDO_ETYPE value of 1003, indicating an exterior polygon ring.
- The nonzero element has an SDO_STARTING_OFFSET value of 11 because ordinate x6 is the eleventh ordinate in the geometry.
- The type 0 element has an SDO_INTERPRETATION value whose significance is application-specific. In this example, the SDO_INTERPRETATION value is 57.

- The nonzero element has an SDO_INTERPRETATION value that is valid for the SDO_ETYPE of 1003. In this example, the SDO_INTERPRETATION value is 3, indicating a rectangle defined by two points (lower-left and upper-right).

[Example 2–12](#) shows a SQL statement that inserts the geometry with a type 0 element (similar to the geometry illustrated in [Figure 2–8](#)) into the database. In the SDO_ORDINATE_ARRAY structure, the curve is defined by points (6,6), (12,6), (9,8), (6,10), and (12,10), and the rectangle is defined by points (6,4) and (12,12).

Example 2–12 SQL Statement to Insert a Geometry with a Type 0 Element

```
INSERT INTO cola_markets VALUES(
  13,
  'type_zero_element_geom',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,0,57, 11,1003,3), -- 1st is type 0 element
    SDO_ORDINATE_ARRAY(6,6, 12,6, 9,8, 6,10, 12,10, 6,4, 12,12)
  )
);
```

2.5.8 Several Geometry Types

[Example 2–13](#) creates a table and inserts various geometries, including multipoints (point clusters), multipolygons, and collections. At the end, it calls the [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#) function to validate the inserted geometries. Note that some geometries are deliberately invalid, and their descriptions include the string INVALID.

Example 2–13 SQL Statements to Insert Various Geometries

```
CREATE TABLE t1 (
  i NUMBER,
  d VARCHAR2(50),
  g SDO_GEOMETRY
);
INSERT INTO t1 (i, d, g)
VALUES (
  1,
  'Point',
  sdo_geometry (2001, null, null, sdo_elem_info_array (1,1,1),
    sdo_ordinate_array (10,5))
);
INSERT INTO t1 (i, d, g)
VALUES (
  2,
  'Line segment',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,2,1),
    sdo_ordinate_array (10,10, 20,10))
);
INSERT INTO t1 (i, d, g)
VALUES (
  3,
  'Arc segment',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,2,2),
    sdo_ordinate_array (10,15, 15,20, 20,15))
);
INSERT INTO t1 (i, d, g)
```

```

VALUES (
  4,
  'Line string',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,2,1),
    sdo_ordinate_array (10,25, 20,30, 25,25, 30,30))
);
INSERT INTO t1 (i, d, g)
VALUES (
  5,
  'Arc string',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,2,2),
    sdo_ordinate_array (10,35, 15,40, 20,35, 25,30, 30,35))
);
INSERT INTO t1 (i, d, g)
VALUES (
  6,
  'Compound line string',
  sdo_geometry (2002, null, null,
    sdo_elem_info_array (1,4,3, 1,2,1, 3,2,2, 7,2,1),
    sdo_ordinate_array (10,45, 20,45, 23,48, 20,51, 10,51))
);
INSERT INTO t1 (i, d, g)
VALUES (
  7,
  'Closed line string',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,2,1),
    sdo_ordinate_array (10,55, 15,55, 20,60, 10,60, 10,55))
);
INSERT INTO t1 (i, d, g)
VALUES (
  8,
  'Closed arc string',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,2,2),
    sdo_ordinate_array (15,65, 10,68, 15,70, 20,68, 15,65))
);
INSERT INTO t1 (i, d, g)
VALUES (
  9,
  'Closed mixed line',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,4,2, 1,2,1, 7,2,2),
    sdo_ordinate_array (10,78, 10,75, 20,75, 20,78, 15,80, 10,78))
);
INSERT INTO t1 (i, d, g)
VALUES (
  10,
  'Self-crossing line',
  sdo_geometry (2002, null, null, sdo_elem_info_array (1,2,1),
    sdo_ordinate_array (10,85, 20,90, 20,85, 10,90, 10,85))
);
INSERT INTO t1 (i, d, g)
VALUES (
  11,
  'Polygon',
  sdo_geometry (2003, null, null, sdo_elem_info_array (1,1003,1),
    sdo_ordinate_array (10,105, 15,105, 20,110, 10,110, 10,105))
);
INSERT INTO t1 (i, d, g)
VALUES (
  12,
  'Arc polygon',

```



```

        sdo_geometry (2003, null, null, sdo_elem_info_array (1,1003,2),
            sdo_ordinate_array (15,115, 20,118, 15,120, 10,118, 15,115))
    );
INSERT INTO t1 (i, d, g)
VALUES (
    13,
    'Compound polygon',
    sdo_geometry (2003, null, null, sdo_elem_info_array (1,1005,2, 1,2,1, 7,2,2),
        sdo_ordinate_array (10,128, 10,125, 20,125, 20,128, 15,130, 10,128))
);
INSERT INTO t1 (i, d, g)
VALUES (
    14,
    'Rectangle',
    sdo_geometry (2003, null, null, sdo_elem_info_array (1,1003,3),
        sdo_ordinate_array (10,135, 20,140))
);
INSERT INTO t1 (i, d, g)
VALUES (
    15,
    'Circle',
    sdo_geometry (2003, null, null, sdo_elem_info_array (1,1003,4),
        sdo_ordinate_array (15,145, 10,150, 20,150))
);
INSERT INTO t1 (i, d, g)
VALUES (
    16,
    'Point cluster',
    sdo_geometry (2005, null, null, sdo_elem_info_array (1,1,3),
        sdo_ordinate_array (50,5, 55,7, 60,5))
);
INSERT INTO t1 (i, d, g)
VALUES (
    17,
    'Multipoint',
    sdo_geometry (2005, null, null, sdo_elem_info_array (1,1,1, 3,1,1, 5,1,1),
        sdo_ordinate_array (65,5, 70,7, 75,5))
);
INSERT INTO t1 (i, d, g)
VALUES (
    18,
    'Multiline',
    sdo_geometry (2006, null, null, sdo_elem_info_array (1,2,1, 5,2,1),
        sdo_ordinate_array (50,15, 55,15, 60,15, 65,15))
);
INSERT INTO t1 (i, d, g)
VALUES (
    19,
    'Multiline - crossing',
    sdo_geometry (2006, null, null, sdo_elem_info_array (1,2,1, 5,2,1),
        sdo_ordinate_array (50,22, 60,22, 55,20, 55,25))
);
INSERT INTO t1 (i, d, g)
VALUES (
    20,
    'Multiarc',
    sdo_geometry (2006, null, null, sdo_elem_info_array (1,2,2, 7,2,2),
        sdo_ordinate_array (50,35, 55,40, 60,35, 65,35, 70,30, 75,35))
);
INSERT INTO t1 (i, d, g)

```

```

VALUES (
  21,
  'Multiline - closed',
  sdo_geometry (2006, null, null, sdo_elem_info_array (1,2,1, 9,2,1),
    sdo_ordinate_array (50,55, 50,60, 55,58, 50,55, 56,58, 60,55, 60,60, 56,58))
);
INSERT INTO t1 (i, d, g)
VALUES (
  22,
  'Multiarc - touching',
  sdo_geometry (2006, null, null, sdo_elem_info_array (1,2,2, 7,2,2),
    sdo_ordinate_array (50,65, 50,70, 55,68, 55,68, 60,65, 60,70))
);
INSERT INTO t1 (i, d, g)
VALUES (
  23,
  'Multipolygon - disjoint',
  sdo_geometry (2007, null, null, sdo_elem_info_array (1,1003,1, 11,1003,3),
    sdo_ordinate_array (50,105, 55,105, 60,110, 50,110, 50,105, 62,108, 65,112))
);
INSERT INTO t1 (i, d, g)
VALUES (
  24,
  'Multipolygon - touching',
  sdo_geometry (2007, null, null, sdo_elem_info_array (1,1003,3, 5,1003,3),
    sdo_ordinate_array (50,115, 55,120, 55,120, 58,122))
);
INSERT INTO t1 (i, d, g)
VALUES (
  25,
  'Multipolygon - tangent * INVALID 13351',
  sdo_geometry (2007, null, null, sdo_elem_info_array (1,1003,3, 5,1003,3),
    sdo_ordinate_array (50,125, 55,130, 55,128, 60,132))
);
INSERT INTO t1 (i, d, g)
VALUES (
  26,
  'Multipolygon - multi-touch',
  sdo_geometry (2007, null, null, sdo_elem_info_array (1,1003,1, 17,1003,1),
    sdo_ordinate_array (50,95, 55,95, 53,96, 55,97, 53,98, 55,99, 50,99, 50,95,
      55,100, 55,95, 60,95, 60,100, 55,100))
);
INSERT INTO t1 (i, d, g)
VALUES (
  27,
  'Polygon with void',
  sdo_geometry (2003, null, null, sdo_elem_info_array (1,1003,3, 5,2003,3),
    sdo_ordinate_array (50,135, 60,140, 51,136, 59,139))
);
INSERT INTO t1 (i, d, g)
VALUES (
  28,
  'Polygon with void - reverse',
  sdo_geometry (2003, null, null, sdo_elem_info_array (1,2003,3, 5,1003,3),
    sdo_ordinate_array (51,146, 59,149, 50,145, 60,150))
);
INSERT INTO t1 (i, d, g)
VALUES (
  29,
  'Crescent (straight lines) * INVALID 13349',

```

```

sdo_geometry (2003, null, null, sdo_elem_info_array (1,1003,1),
  sdo_ordinate_array (10,175, 10,165, 20,165, 15,170, 25,170, 20,165,
    30,165, 30,175, 10,175))
);
INSERT INTO t1 (i, d, g)
VALUES (
  30,
  'Crescent (arcs) * INVALID 13349',
  sdo_geometry (2003, null, null, sdo_elem_info_array (1,1003,2),
    sdo_ordinate_array (14,180, 10,184, 14,188, 18,184, 14,180, 16,182,
      14,184, 12,182, 14,180))
);
INSERT INTO t1 (i, d, g)
VALUES (
  31,
  'Heterogeneous collection',
  sdo_geometry (2004, null, null, sdo_elem_info_array (1,1,1, 3,2,1, 7,1003,1),
    sdo_ordinate_array (10,5, 10,10, 20,10, 10,105, 15,105, 20,110, 10,110,
      10,105))
);
INSERT INTO t1 (i, d, g)
VALUES (
  32,
  'Polygon+void+island touch',
  sdo_geometry (2007, null, null,
    sdo_elem_info_array (1,1003,1, 11,2003,1, 31,1003,1),
    sdo_ordinate_array (50,168, 50,160, 55,160, 55,168, 50,168, 51,167,
      54,167, 54,161, 51,161, 51,162, 52,163, 51,164, 51,165, 51,166, 51,167,
      52,166, 52,162, 53,162, 53,166, 52,166))
);
COMMIT;
SELECT i, d, SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT (g, 0.5) FROM t1;

```

2.6 Geometry Metadata Views

The geometry metadata describing the dimensions, lower and upper bounds, and tolerance in each dimension is stored in a global table owned by MDSYS (which users should never directly update). Each Spatial user has the following views available in the schema associated with that user:

- **USER_SDO_GEOM_METADATA** contains metadata information for all spatial tables owned by the user (schema). This is the only view that you can update, and it is the one in which Spatial users must insert metadata related to spatial tables.
- **ALL_SDO_GEOM_METADATA** contains metadata information for all spatial tables on which the user has SELECT permission.

Spatial users are responsible for populating these views. For each spatial column, you must insert an appropriate row into the **USER_SDO_GEOM_METADATA** view. Oracle Spatial ensures that the **ALL_SDO_GEOM_METADATA** view is also updated to reflect the rows that you insert into **USER_SDO_GEOM_METADATA**.

Note: These views were new for release 8.1.6. If you are upgrading from an earlier release of Spatial, see [Appendix A](#) and the information about the [SDO_MIGRATE.TO_CURRENT](#) procedure in [Chapter 17](#).

Each metadata view has the following definition:

```
(
  TABLE_NAME  VARCHAR2(32),
  COLUMN_NAME  VARCHAR2(32),
  DIMINFO      SDO_DIM_ARRAY,
  SRID         NUMBER
);
```

In addition, the ALL_SDO_GEOM_METADATA view has an OWNER column identifying the schema that owns the table specified in TABLE_NAME.

The following considerations apply to schema, table, and column names that are stored in any Oracle Spatial metadata views:

- The name must contain only letters, numbers, and underscores. For example, the name cannot contain a space (), an apostrophe ('), a quotation mark ("), or a comma (,).
- All letters in the names are converted to uppercase before the names are stored in geometry metadata views or before the tables are accessed. This conversion also applies to any schema name specified with the table name.

2.6.1 TABLE_NAME

The TABLE_NAME column contains the name of a feature table, such as COLA_MARKETS, that has a column of type SDO_GEOMETRY.

The table name is stored in the spatial metadata views in all uppercase characters.

The table name cannot contain spaces or mixed-case letters in a quoted string when inserted into the USER_SDO_GEOM_METADATA view, and it cannot be in a quoted string when used in a query (unless it is in all uppercase characters).

The spatial feature table cannot be an index-organized table if you plan to create a spatial index on the spatial column.

2.6.2 COLUMN_NAME

The COLUMN_NAME column contains the name of the column of type SDO_GEOMETRY. For the COLA_MARKETS table, this column is called SHAPE.

The column name is stored in the spatial metadata views in all uppercase characters.

The column name cannot contain spaces or mixed-case letters in a quoted string when inserted into the USER_SDO_GEOM_METADATA view, and it cannot be in a quoted string when used in a query (unless it is in all uppercase characters).

2.6.3 DIMINFO

The DIMINFO column is a varying length array of an object type, ordered by dimension, and has one entry for each dimension. The SDO_DIM_ARRAY type is defined as follows:

```
Create Type SDO_DIM_ARRAY as VARRAY(4) of SDO_DIM_ELEMENT;
```

The SDO_DIM_ELEMENT type is defined as:

```
Create Type SDO_DIM_ELEMENT as OBJECT (
  SDO_DIMNAME VARCHAR2(64),
  SDO_LB NUMBER,
  SDO_UB NUMBER,
  SDO_TOLERANCE NUMBER);
```

The SDO_DIM_ARRAY instance is of size n if there are n dimensions. That is, DIMINFO contains 2 SDO_DIM_ELEMENT instances for two-dimensional geometries, 3 instances for three-dimensional geometries, and 4 instances for four-dimensional geometries. Each SDO_DIM_ELEMENT instance in the array must have valid (not null) values for the SDO_LB, SDO_UB, and SDO_TOLERANCE attributes.

Note: The number of dimensions reflected in the DIMINFO information must match the number of dimensions of each geometry object in the layer.

For an explanation of tolerance and how to determine the appropriate SDO_TOLERANCE value, see [Section 1.5.5](#), especially [Section 1.5.5.1](#).

Spatial assumes that the varying length array is ordered by dimension. The DIMINFO varying length array must be ordered by dimension in the same way the ordinates for the points in SDO_ORDINATES varying length array are ordered. For example, if the SDO_ORDINATES varying length array contains $\{X1, Y1, \dots, Xn, Yn\}$, then the first DIMINFO entry must define the X dimension and the second DIMINFO entry must define the Y dimension.

[Example 2–1](#) in [Section 2.1](#) shows the use of the SDO_GEOMETRY and SDO_DIM_ARRAY types. This example demonstrates how geometry objects (hypothetical market areas for colas) are represented, and how the COLA_MARKETS feature table and the USER_SDO_GEOM_METADATA view are populated with the data for those objects.

2.6.4 SRID

The SRID column should contain either of the following: the SRID value for the coordinate system for all geometries in the column, or NULL if no specific coordinate system should be associated with the geometries. (For information about coordinate systems, see [Chapter 6](#).)

2.7 Spatial Index-Related Structures

This section describes the structure of the tables containing the spatial index data and metadata. Concepts and usage notes for spatial indexing are explained in [Section 1.7](#). The spatial index data and metadata are stored in tables that are created and maintained by the Spatial indexing routines. These tables are created in the schema of the owner of the feature (underlying) table that has a spatial index created on a column of type SDO_GEOMETRY.

2.7.1 Spatial Index Views

There are two sets of spatial index metadata views for each schema (user): `xxx_SDO_INDEX_INFO` and `xxx_SDO_INDEX_METADATA`, where `xxx` can be `USER` or `ALL`. These views are read-only to users; they are created and maintained by the Spatial indexing routines.

2.7.1.1 xxx_SDO_INDEX_INFO Views

The following views contain basic information about spatial indexes:

- `USER_SDO_INDEX_INFO` contains index information for all spatial tables owned by the user.

- ALL_SDO_INDEX_INFO contains index information for all spatial tables on which the user has SELECT permission.

The USER_SDO_INDEX_INFO and ALL_SDO_INDEX_INFO views contain the same columns, as shown [Table 2-4](#), except that the USER_SDO_INDEX_INFO view does not contain the SDO_INDEX_OWNER column. (The columns are listed in their order in the view definition.)

Table 2-4 Columns in the xxx_SDO_INDEX_INFO Views

Column Name	Data Type	Purpose
SDO_INDEX_OWNER	VARCHAR2	Owner of the index (ALL_SDO_INDEX_INFO views only).
INDEX_NAME	VARCHAR2	Name of the index.
TABLE_NAME	VARCHAR2	Name of the table containing the column on which this index is built.
COLUMN_NAME	VARCHAR2	Name of the column on which this index is built.
SDO_INDEX_TYPE	VARCHAR2	Contains QTREE (for a quadtree index) or RTREE (for an R-tree index).
SDO_INDEX_TABLE	VARCHAR2	Name of the spatial index table (described in Section 2.7.2).
SDO_INDEX_STATUS	VARCHAR2	(Deprecated; reserved for Oracle use.)

2.7.1.2 xxx_SDO_INDEX_METADATA Views

The following views contain detailed information about spatial index metadata:

- USER_SDO_INDEX_METADATA contains index information for all spatial tables owned by the user. (USER_SDO_INDEX_METADATA is the same as SDO_INDEX_METADATA, which was the only metadata view for Oracle Spatial release 8.1.5.)
- ALL_SDO_INDEX_METADATA contains index information for all spatial tables on which the user has SELECT permission.

Note: These views were new for release 8.1.6. If you are upgrading from an earlier release of Spatial, see [Appendix A](#).

The USER_SDO_INDEX_METADATA and ALL_SDO_INDEX_METADATA views contain the same columns, as shown [Table 2-5](#). (The columns are listed in their order in the view definition.)

Table 2-5 Columns in the xxx_SDO_INDEX_METADATA Views

Column Name	Data Type	Purpose
SDO_INDEX_OWNER	VARCHAR2	Owner of the index.
SDO_INDEX_TYPE	VARCHAR2	Contains QTREE (for a quadtree index) or RTREE (for an R-tree index).
SDO_LEVEL	NUMBER	The fixed tiling level at which to tile all objects in the geometry column for a quadtree index.
SDO_NUMTILES	NUMBER	Suggested number of tiles per object that should be used to approximate the shape for a quadtree index.

Table 2–5 (Cont.) Columns in the xxx_SDO_INDEX_METADATA Views

Column Name	Data Type	Purpose
SDO_MAXLEVEL	NUMBER	Maximum level for any tile for any object for a quadtree index. It will always be greater than the SDO_LEVEL value.
SDO_COMMIT_INTERVAL	NUMBER	Number of geometries (rows) to process, during index creation, before committing the insertion of spatial index entries into the SDOINDEX table. (Applies to quadtree indexes only.)
SDO_INDEX_TABLE	VARCHAR2	Name of the spatial index table (described in Section 2.7.2).
SDO_INDEX_NAME	VARCHAR2	Name of the index.
SDO_INDEX_PRIMARY	NUMBER	Indicates if this is a primary or secondary index. 1 = primary, 2 = secondary.
SDO_TSNAME	VARCHAR2	Schema name of the SDO_INDEX_TABLE.
SDO_COLUMN_NAME	VARCHAR2	Name of the column on which this index is built.
SDO_RTREE_HEIGHT	NUMBER	Height of the R-tree.
SDO_RTREE_NUM_NODES	NUMBER	Number of nodes in the R-tree.
SDO_RTREE_DIMENSIONALITY	NUMBER	Number of dimensions used internally by Spatial. This may be different from the number of dimensions indexed, which is controlled by the <code>sdo_indx_dims</code> keyword in the CREATE INDEX or ALTER INDEX statement, and which is stored in the SDO_INDEX_DIMS column in this view. For example, for an index on geodetic data, the SDO_RTREE_DIMENSIONALITY value is 3, but the SDO_INDEX_DIMS value is 2.
SDO_RTREE_FANOUT	NUMBER	Maximum number of children in each R-tree node.
SDO_RTREE_ROOT	VARCHAR2	Rowid corresponding to the root node of the R-tree in the index table.
SDO_RTREE_SEQ_NAME	VARCHAR2	Sequence name associated with the R-tree.
SDO_FIXED_META	RAW	If applicable, this column contains the metadata portion of the SDO_GROUPCODE or SDO_CODE for a fixed-level index.
SDO_TABLESPACE	VARCHAR2	Same as in the SQL CREATE TABLE statement. Tablespace in which to create the SDOINDEX table.
SDO_INITIAL_EXTENT	VARCHAR2	Same as in the SQL CREATE TABLE statement.
SDO_NEXT_EXTENT	VARCHAR2	Same as in the SQL CREATE TABLE statement.
SDO_PCTINCREASE	NUMBER	Same as in the SQL CREATE TABLE statement.
SDO_MIN_EXTENTS	NUMBER	Same as in the SQL CREATE TABLE statement.
SDO_MAX_EXTENTS	NUMBER	Same as in the SQL CREATE TABLE statement.
SDO_INDEX_DIMS	NUMBER	Number of dimensions of the geometry objects in the column on which this index is built, as determined by the value of the <code>sdo_indx_dims</code> keyword in the CREATE INDEX or ALTER INDEX statement.

Table 2–5 (Cont.) Columns in the xxx_SDO_INDEX_METADATA Views

Column Name	Data Type	Purpose
SDO_LAYER_GTYPE	VARCHAR2	Contains DEFAULT if the layer can contain both point and polygon data, or a value from the Geometry Type column of Table 2–1 in Section 2.2.1 .
SDO_RTREE_PCTFREE	NUMBER	Minimum percentage of slots in each index tree node to be left empty when an R-tree index is created.
SDO_INDEX_PARTITION	VARCHAR2	For a partitioned index, name of the index partition.
SDO_PARTITIONED	NUMBER	Contains 0 if the index is not partitioned or 1 if the index is partitioned.
SDO_RTREE_QUALITY	NUMBER	Quality score for an index. See the information about R-tree quality in Section 1.7.2 .
SDO_INDEX_VERSION	NUMBER	Internal version number of the index.
SDO_INDEX_GEODETTIC	VARCHAR2	Contains TRUE if the index is geodetic (see Section 4.1.1) and FALSE if the index is not geodetic.
SDO_INDEX_STATUS	VARCHAR2	(Deprecated; reserved for Oracle use.)
SDO_NL_INDEX_TABLE	VARCHAR2	Name of a separate index table (with a name in the form MDNT_...\$) for nonleaf nodes of the index. For more information, see the description of the <code>sdo_non_leaf_tbl</code> parameter for the CREATE INDEX statement in Chapter 10 .
SDO_DML_BATCH_SIZE	NUMBER	Number of index updates to be processed in each batch of updates after a commit operation. For more information, see the description of the <code>sdo_dml_batch_size</code> parameter for the CREATE INDEX statement in Chapter 10 .
SDO_RTREE_EXT_XPND	NUMBER	(Reserved for future use.)
SDO_ROOT_MBR	SDO_GEOMETRY	Minimum bounding rectangle of the maximum extent of the spatial layer. This is greater than or equal to the MBR of the current extent, and is reset to reflect the current extent when the index is rebuilt.

2.7.2 Spatial Index Table Definition

For an R-tree index, a spatial index table (each SDO_INDEX_TABLE entry as described in [Table 2–5](#) in [Section 2.7.1.2](#)) contains the columns shown in [Table 2–6](#).

Table 2–6 Columns in an R-Tree Spatial Index Data Table

Column Name	Data Type	Purpose
NODE_ID	NUMBER	Unique ID number for this node of the tree.
NODE_LEVEL	NUMBER	Level of the node in the tree. Leaf nodes (nodes whose entries point to data items in the base table) are at level 1, their parent nodes are at level 2, and so on.
INFO	BLOB	Other information in a node. Includes an array of <child_mbr, child_rowid> pairs (maximum of fanout value, or number of children for such pairs in each R-tree node), where child_rowid is the rowid of a child node, or the rowid of a data item from the base table.

2.7.3 R-Tree Index Sequence Object

Each R-tree spatial index table has an associated sequence object (SDO_RTREE_SEQ_NAME in the USER_SDO_INDEX_METADATA view, described in [Table 2-5](#) in [Section 2.7.1.2](#)). The sequence is used to ensure that simultaneous updates can be performed to the index by multiple concurrent users.

The sequence name is the index table name with the letter *S* replacing the letter *T* before the underscore (for example, the sequence object MDRS_5C01\$ is associated with the index table MDRT_5C01\$).

2.8 Unit of Measurement Support

Geometry functions that involve measurement allow an optional `unit` parameter to specify the unit of measurement for a specified distance or area, if a georeferenced coordinate system (SDO_SRID value) is associated with the input geometry or geometries. The `unit` parameter is not valid for geometries with a null SDO_SRID value (that is, an orthogonal Cartesian system). For information about support for coordinate systems, see [Chapter 6](#).

The default unit of measure is the one associated with the georeferenced coordinate system. The unit of measure for most coordinate systems is the meter, and in these cases the default unit for distances is meter and the default unit for areas is square meter. By using the `unit` parameter, however, you can have Spatial automatically convert and return results that are more meaningful to application users, for example, displaying the distance to a restaurant in miles.

The `unit` parameter must be enclosed in single quotation marks and contain the string `unit=` and a valid UNIT_OF_MEAS_NAME value from the SDO_UNITS_OF_MEASURE table (described in [Section 6.6.27](#)). For example, `'unit=KM'` in the following example (using data and definitions from [Example 6-7](#) in [Section 6.11](#)) specifies kilometers as the unit of measurement:

```
SELECT c.name, SDO_GEOM.SDO_LENGTH(c.shape, m.diminfo, 'unit=KM')
   FROM cola_markets_cs c, user_sdo_geom_metadata m
   WHERE m.table_name = 'COLA_MARKETS_CS' AND m.column_name = 'SHAPE';
```

Spatial uses the information in the SDO_UNITS_OF_MEASURE table (described in [Section 6.6.27](#)) to determine which unit names are valid and what ratios to use in comparing or converting between different units. For convenience, you can also use the following legacy views to see the angle, area, and distance units of measure:

- MDSYS.SDO_ANGLE_UNITS (described in [Section 6.7.2](#))
- MDSYS.SSDO_AREA_UNITS (described in [Section 6.7.3](#))
- MDSYS.SSDO_DIST_UNITS (described in [Section 6.7.5](#))

Loading Spatial Data

This chapter describes how to load spatial data into a database, including storing the data in a table with a column of type SDO_GEOMETRY. After you have loaded spatial data, you can create a spatial index for it and perform queries on it, as described in [Chapter 4](#).

The process of loading data can be classified into two categories:

- Bulk loading of data (see [Section 3.1](#))
This process is used to load large volumes of data into the database and uses the SQL*Loader utility to load the data.
- Transactional insert operations (see [Section 3.2](#))
This process is typically used to insert relatively small amounts of data into the database using the INSERT statement in SQL.

3.1 Bulk Loading

Bulk loading can import large amounts of data into an Oracle database. Bulk loading is accomplished with the SQL*Loader utility. (For information about SQL*Loader, see *Oracle Database Utilities*.)

3.1.1 Bulk Loading SDO_GEOMETRY Objects

[Example 3-1](#) is the SQL*Loader control file for loading four geometries. When this control file is used with SQL*Loader, it loads the same cola market geometries that are inserted using SQL statements in [Example 2-1](#) in [Section 2.1](#).

Example 3-1 Control File for a Bulk Load of Cola Market Geometries

```
LOAD DATA
INFILE *
TRUNCATE
CONTINUEIF NEXT(1:1) = '#'
INTO TABLE COLA_MARKETS
FIELDS TERMINATED BY '|'
TRAILING NULLCOLS (
mkt_id INTEGER EXTERNAL,
name CHAR,
shape COLUMN OBJECT
(
SDO_GTYPE INTEGER EXTERNAL,
SDO_ELEM_INFO VARRAY TERMINATED BY '|/'
(elements FLOAT EXTERNAL),
```

```

SDO_ORDINATES VARRAY TERMINATED BY '|/'
(ordinates FLOAT EXTERNAL)
)
)
begindata
 1|cola_a|
#2003|1|1003|3|/
#1|1|5|7|/
 2|cola_b|
#2003|1|1003|1|/
#5|1|8|1|8|6|5|7|5|1|/
 3|cola_c|
#2003|1|1003|1|/
#3|3|6|3|6|5|4|5|3|3|/
 4|cola_d|
#2003|1|1003|4|/
#8|7|10|9|8|11|/

```

Notes on [Example 3-1](#):

- The EXTERNAL keyword in the definition `mkt_id INTEGER EXTERNAL` means that each value to be inserted into the MKT_ID column (1, 2, 3, and 4 in this example) is an integer in human-readable form, not binary format.
- In the data after `begindata`, each MKT_ID value is preceded by one space, because the `CONTINUEIF NEXT(1:1) = '#'` specification causes the first position of each data line to be ignored unless it is the number sign (#) continuation character.

[Example 3-2](#) assumes that a table named POLY_4PT was created as follows:

```

CREATE TABLE POLY_4PT (GID      VARCHAR2(32),
                       GEOMETRY  SDO_GEOMETRY);

```

Assume that the ASCII data consists of a file with delimited columns and separate rows fixed by the limits of the table with the following format:

```

geometry rows:      GID, GEOMETRY

```

The coordinates in the GEOMETRY column represent polygons. [Example 3-2](#) shows the control file for loading the data.

Example 3-2 Control File for a Bulk Load of Polygons

```

LOAD DATA
  INFILE *
  TRUNCATE
  CONTINUEIF NEXT(1:1) = '#'
  INTO TABLE POLY_4PT
  FIELDS TERMINATED BY '|/'
  TRAILING NULLCOLS (
    GID  INTEGER EXTERNAL,
    GEOMETRY COLUMN OBJECT
  (
    SDO_GTYPE      INTEGER EXTERNAL,
    SDO_ELEM_INFO  VARRAY TERMINATED BY '|/'
                   (elements  FLOAT EXTERNAL),
    SDO_ORDINATES  VARRAY TERMINATED BY '|/'
                   (ordinates  FLOAT EXTERNAL)
  )
)
begindata

```

```

1|2003|1|1003|1|/
#-122.4215|37.7862|-122.422|37.7869|-122.421|37.789|-122.42|37.7866|
#-122.4215|37.7862|/
2|2003|1|1003|1|/
#-122.4019|37.8052|-122.4027|37.8055|-122.4031|37.806|-122.4012|37.8052|
#-122.4019|37.8052|/
3|2003|1|1003|1|/
#-122.426|37.803|-122.4242|37.8053|-122.42355|37.8044|-122.4235|37.8025|
#-122.426|37.803|/

```

3.1.2 Bulk Loading Point-Only Data in SDO_GEOMETRY Objects

[Example 3-3](#) shows a control file for loading a table with point data.

Example 3-3 Control File for a Bulk Load of Point-Only Data

```

LOAD DATA
  INFILE *
  TRUNCATE
  CONTINUEIF NEXT(1:1) = '#'
  INTO TABLE POINT
  FIELDS TERMINATED BY '|'
  TRAILING NULLCOLS (
    GID      INTEGER EXTERNAL,
    GEOMETRY COLUMN OBJECT
    (
      SDO_GTYPE      INTEGER EXTERNAL,
      SDO_POINT COLUMN OBJECT
      (X      FLOAT EXTERNAL,
       Y      FLOAT EXTERNAL)
    )
  )

BEGINDATA
1|
200
1| -122.4215| 37.7862|
2|
200
1| -122.4019| 37.8052|
3|
200
1| -122.426| 37.803|
4|
200
1| -122.4171| 37.8034|
5|
200
1| -122.416151| 37.8027228|

```

3.2 Transactional Insert Operations Using SQL

Oracle Spatial uses standard Oracle tables that can be accessed or loaded with standard SQL syntax. This section contains examples of transactional insertions into columns of type SDO_GEOMETRY. This process is typically used to add relatively small amounts of data into the database.

The INSERT statement in Oracle SQL has a limit of 999 arguments. Therefore, you cannot create a variable-length array of more than 999 elements using the SDO_

GEOMETRY constructor inside a transactional INSERT statement; however, you can insert a geometry using a host variable, and the host variable can be built using the SDO_GEOMETRY constructor with more than 999 values in the SDO_ORDINATE_ARRAY specification. (The host variable is an OCI, PL/SQL, or Java program variable.)

To perform transactional insertions of geometries, you can create a procedure to insert a geometry, and then invoke that procedure on each geometry to be inserted.

[Example 3-4](#) creates a procedure to perform the insert operation.

Example 3-4 Procedure to Perform a Transactional Insert Operation

```
CREATE OR REPLACE PROCEDURE
    INSERT_GEOM(GEOM SDO_GEOMETRY)
IS
BEGIN
    INSERT INTO TEST_1 VALUES (GEOM);
    COMMIT;
END;
/
```

Using the procedure created in [Example 3-4](#), you can insert data by using a PL/SQL block, such as the one in [Example 3-5](#), which loads a geometry into the variable named `geom` and then invokes the `INSERT_GEOM` procedure to insert that geometry.

Example 3-5 PL/SQL Block Invoking a Procedure to Insert a Geometry

```
DECLARE
geom SDO_geometry :=
    SDO_geometry (2003, null, null,
        SDO_elem_info_array (1,1003,3),
        SDO_ordinate_array (-109,37,-102,40));
BEGIN
    INSERT_GEOM(geom);
    COMMIT;
END;
/
```

For additional examples with various geometry types, see the following:

- Rectangle: [Example 2-4](#) in [Section 2.5.1](#)
- Polygon with a hole: [Example 2-5](#) in [Section 2.5.2](#)
- Compound line string: [Example 2-6](#) in [Section 2.5.3](#)
- Compound polygon: [Example 2-7](#) in [Section 2.5.4](#)
- Point: [Example 2-8](#) and [Example 2-9](#) in [Section 2.5.5](#)
- Oriented point: [Example 2-10](#) in [Section 2.5.6](#)
- Type 0 (zero) element: [Example 2-12](#) in [Section 2.5.7](#)

Indexing and Querying Spatial Data

After you have loaded spatial data (discussed in [Chapter 3](#)), you should create a spatial index on it to enable efficient query performance using the data. This chapter describes how to:

- Create a spatial index (see [Section 4.1](#))
- Query spatial data efficiently, based on an understanding of the Oracle Spatial query model and primary and secondary filtering (see [Section 4.2](#))

4.1 Creating a Spatial Index

Once data has been loaded into the spatial tables through either bulk or transactional loading, a spatial index (that is, a spatial R-tree index) must be created on the tables for efficient access to the data. For example, the following statement creates a spatial index named `territory_idx` using default values for all parameters:

```
CREATE INDEX territory_idx ON territories (territory_geom)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

For detailed information about options for creating a spatial index, see the documentation for the [CREATE INDEX](#) statement in [Chapter 10](#).

If the index creation does not complete for any reason, the index is invalid and must be deleted with the [DROP INDEX <index_name> \[FORCE\]](#) statement.

Spatial indexes can be built on two, three, or four dimensions of data. The default number of dimensions is two, but if the data has more than two dimensions, you can use the `sdo_indx_dims` parameter keyword to specify the number of dimensions on which to build the index. However, if a spatial index has been built on more than two dimensions of a layer, the only spatial operator that can be used against that layer is [SDO_FILTER](#) (the primary filter or index-only query), which considers all dimensions. The [SDO_RELATE](#), [SDO_NN](#), and [SDO_WITHIN_DISTANCE](#) operators are disabled if the index has been built on more than two dimensions.

If the rollback segment is not large enough, an attempt to create a spatial index will fail. The rollback segment should be $100*n$ bytes, where n is the number of rows of data to be indexed. For example, if the table contains 1 million (1,000,000) rows, the rollback segment size should be 100,000,000 (100 million) bytes.

To ensure an adequate rollback segment, or if you have tried to create a spatial index and received an error that a rollback segment cannot be extended, review (or have a DBA review) the size and structure of the rollback segments. Create a public rollback segment of the appropriate size, and place that rollback segment online. In addition, ensure that any small inappropriate rollback segments are placed offline during large

spatial index operations. For information about performing these operations on a rollback segment, see *Oracle Database Administrator's Guide*.

The system parameter `SORT_AREA_SIZE` affects the amount of time required to create the index. The `SORT_AREA_SIZE` value is the maximum amount, in bytes, of memory to use for a sort operation. The optimal value depends on the database size, but a good guideline is to make it at least 1 million bytes when you create a spatial index. To change the `SORT_AREA_SIZE` value, use the `ALTER SESSION` statement. For example, to change the value to 20 million bytes:

```
ALTER SESSION SET SORT_AREA_SIZE = 20000000;
```

The tablespace specified with the `tablespace` keyword in the `CREATE INDEX` statement (or the default tablespace if the `tablespace` keyword is not specified) is used to hold both the index data table and some transient tables that are created for internal computations. If you specify `WORK_TABLESPACE` as the tablespace, the transient tables are stored in the work tablespace.

For large tables (over 1 million rows), a temporary tablespace may be needed to perform internal sorting operations. The recommended size for this temporary tablespace is $100*n$ bytes, where n is the number of rows in the table, up to a maximum requirement of 1 gigabyte of temporary tablespace.

To estimate the space that will be needed to create a spatial index, use the `SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE` function, described in [Chapter 19](#).

4.1.1 Indexing Geodetic Data

To take full advantage of Spatial features, you must index geodetic data using a geodetic R-tree index. *Geodetic data* consists of geometries that have geodetic `SDO_SRID` values, reflecting the fact that they are based on a geodetic coordinate system (such as using longitude and latitude) as opposed to a flat or projected plane coordinate system. ([Chapter 6](#) explains coordinate systems and related concepts.) A *geodetic index* is one that provides the full range of Spatial features with geodetic data. Thus, it is highly recommended that you use a geodetic index with geodetic data.

Only R-tree indexes can be geodetic indexes. Quadtree indexes cannot be geodetic indexes. If you create an R-tree or quadtree index and specify `'geodetic=false'` in the `CREATE INDEX` statement, the index is non-geodetic. The following notes and restrictions apply to non-geodetic indexes:

- If you create a non-geodetic index on geodetic data, you cannot use the `unit` parameter with the `SDO_WITHIN_DISTANCE` operator or the `SDO_NN_DISTANCE` ancillary operator with the `SDO_NN` operator.
- If you create a non-geodetic index on projected data that has a projected `SDO_SRID` value, you can use the full range of Spatial features.
- If you create a non-geodetic index on projected data that has a null `SDO_SRID` value, you cannot use the `unit` parameter with the `SDO_WITHIN_DISTANCE` operator or the `SDO_NN_DISTANCE` ancillary operator with the `SDO_NN` operator.

For additional information, see the Usage Notes about the `geodetic` parameter for the `CREATE INDEX` statement in [Chapter 10](#).

4.1.2 Constraining Data to a Geometry Type

When you create or rebuild a spatial index, you can ensure that all geometries that are in the table or that are inserted later are of a specified geometry type. To constrain the

data to a geometry type in this way, use the `layer_gtype` keyword in the `PARAMETERS` clause of the `CREATE INDEX` or `ALTER INDEX REBUILD` statement, and specify a value from the Geometry Type column of [Table 2–1](#) in [Section 2.2.1](#). For example, to constrain spatial data in a layer to polygons:

```
CREATE INDEX cola_spatial_idx
ON cola_markets(shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS ('layer_gtype=POLYGON');
```

The geometry types in [Table 2–1](#) are considered as a hierarchy when data is checked:

- The *MULTI* forms include the regular form also. For example, specifying `'layer_gtype=MULTIPOINT'` allows the layer to include both `POINT` and `MULTIPOINT` geometries.
- `COLLECTION` allows the layer to include all types of geometries.

4.1.3 Creating a Cross-Schema Index

You can create a spatial index on a table that is not in your schema. Assume that user B wants to create a spatial index on column `GEOMETRY` in table `T1` under user A's schema. Follow these steps:

1. Connect to the database as a privileged user (for example, as `SYSTEM`), and execute the following statement:

```
GRANT create table, create sequence to B;
```

2. Connect as a privileged user or as user A (or have user A connect), and execute the following statement:

```
GRANT select, index on A.T1 to B;
```

3. Connect as user B and execute a statement such as the following:

```
CREATE INDEX t1_spatial_idx on A.T1(geometry)
INDEXTYPE IS mdsys.spatial_index;
```

4.1.4 Using Partitioned Spatial Indexes

You can create a partitioned spatial index on a partitioned table. This section describes usage considerations specific to Oracle Spatial. For a detailed explanation of partitioned tables and partitioned indexes, see *Oracle Database Administrator's Guide*.

A partitioned spatial index can provide the following benefits:

- Reduced response times for long-running queries, because partitioning reduces disk I/O operations
- Reduced response times for concurrent queries, because I/O operations run concurrently on each partition
- Easier index maintenance, because of partition-level create and rebuild operations

Indexes on partitions can be rebuilt without affecting the queries on other partitions, and storage parameters for each local index can be changed independent of other partitions.
- Parallel query on multiple partition searching

The degree of parallelism is the value from the `DEGREE` column in the row for the index in the `USER_INDEXES` view (that is, the value specified or defaulted for the

PARALLEL keyword with the [CREATE INDEX](#), [ALTER INDEX](#), or [ALTER INDEX REBUILD](#) statement).

- Improved query processing in multiprocessor system environments
 In a multiprocessor system environment, if a spatial operator is invoked on a table with partitioned spatial index and if multiple partitions are involved in the query, multiple processors can be used to evaluate the query. The number of processors used is determined by the degree of parallelism and the number of partitions used in evaluating the query.

The following restrictions apply to spatial index partitioning:

- The partition key for spatial tables must be a scalar value, and must not be a spatial column.
- Only range partitioning is supported on the underlying table. Hash and composite partitioning are not currently supported for partitioned spatial indexes.

To create a partitioned spatial index, you must specify the LOCAL keyword. For example:

```
CREATE INDEX counties_idx ON counties(geometry)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX LOCAL;
```

In this example, the default values are used for the number and placement of index partitions, namely:

- Index partitioning is based on the underlying table partitioning. For each table partition, a corresponding index partition is created.
- Each index partition is placed in the default tablespace.

If you do specify parameters for individual partitions, the following considerations apply:

- The storage characteristics for each partition can be the same or different for each partition. If they are different, it may enable parallel I/O (if the tablespaces are on different disks) and may improve performance.
- The `sdo_indx_dims` value must be the same for all partitions.
- The `layer_gtype` parameter value (see [Section 4.1.2](#)) used for each partition may be different.

To override the default partitioning values, use a CREATE INDEX statement with the following general format:

```
CREATE INDEX <indexname> ON <table>(<column>)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  [PARAMETERS ('<spatial-params>, <storage-params>')] LOCAL
  [( PARTITION <index_partition>
    PARAMETERS ('<spatial-params>, <storage-params>')
  [, PARTITION <index_partition>
    PARAMETERS ('<spatial-params>, <storage-params>')]
  )]
```

Queries can operate on partitioned tables to perform the query on only one partition. For example:

```
SELECT * FROM counties PARTITION(p1)
  WHERE ...<some-spatial-predicate>;
```

Querying on a selected partition may speed up the query and also improve overall throughput when multiple queries operate on different partitions concurrently.

When queries use a partitioned spatial index, the semantics (meaning or behavior) of spatial operators and functions is the same with partitioned and nonpartitioned indexes, except in the case of `SDO_NN` (nearest neighbor). With `SDO_NN`, the requested number of geometries is returned for each partition that is affected by the query. For example, if you request the 5 closest restaurants to a point and the spatial index has 4 partitions, `SDO_NN` returns up to 20 (5*4) geometries. In this case, you must use the `ROWNUM` pseudocolumn (here, `WHERE ROWNUM <=5`) to return the 5 closest restaurants. See the description of the `SDO_NN` operator in [Chapter 11](#) for more information.

4.1.5 Exchanging Partitions Including Indexes

You can use the `ALTER TABLE` statement with the `EXCHANGE PARTITION ... INCLUDING INDEXES` clause to exchange a spatial table partition and its index partition with a corresponding table and its index. For information about exchanging partitions, see the description of the `ALTER TABLE` statement in *Oracle Database SQL Reference*.

This feature can help you to operate more efficiently in a number of situations, such as:

- Bringing data into a partitioned table and avoiding the cost of index re-creation.
- Managing and creating partitioned indexes. For example, the data could be divided into multiple tables. The index for each table could be built one after the other to minimize the memory and tablespace resources needed during index creation. Alternately, the indexes could be created in parallel in multiple sessions. The tables (along with the indexes) could then be exchanged with the partitions of the original data table.
- Managing offline insert operations. New data can be stored in a temporary table and periodically exchanged with a new partition (for example, in a database with historical data).

To exchange partitions including indexes with spatial data and indexes, the two spatial indexes (one on the partition, the other on the table) must be of compatible types. Specifically:

- Both indexes must have the same dimensionality (`sdo_indx_dims` value).
- Both indexes must be either geodetic or non-geodetic. (Geodetic and non-geodetic indexes are explained in [Section 4.1.1](#).)

If the indexes are not compatible, an error is raised. The table data is exchanged, but the indexes are not exchanged and the indexes are marked as failed. To use the indexes, you must rebuild them.

4.1.6 Export and Import Considerations with Spatial Indexes and Data

If you use the Export utility to export tables with spatial data, the behavior of the operation depends on whether or not the spatial data has been spatially indexed:

- If the spatial data has not been spatially indexed, the table data is exported. However, you must update the `USER_SDO_GEOM_METADATA` view with the appropriate information on the target system.
- If the spatial data has been spatially indexed, the table data is exported, the appropriate information is inserted into the `USER_SDO_GEOM_METADATA` view on the target system, and the spatial index is built on the target system.

However, if the insertion into the USER_SDO_GEOM_METADATA view fails (for example, if there is already a USER_SDO_GEOM_METADATA entry for the spatial layer), the spatial index is not built.

If you use the Import utility to import data that has been spatially indexed, the following considerations apply:

- If the index on the exported data was created with a TABLESPACE clause and if the specified tablespace does not exist in the database at import time, the index is not built. (This is different from the behavior with other Oracle indexes, where the index is created in the user's default tablespace if the tablespace specified for the original index does not exist at import time.)
- If the import operation must be done by a privileged database user, and if the FROMUSER and TOUSER format is used, the TOUSER user must be granted the CREATE TABLE and CREATE SEQUENCE privileges before the import operation, as shown in the following example:

```
sqlplus system/<password>
SQL> grant CREATE TABLE, CREATE SEQUENCE to CHRIS;
SQL> exit;
imp system/<password> file=spatl_data.dmp fromuser=SCOTT touser=CHRIS
```

For information about using the Export and Import utilities, see *Oracle Database Utilities*.

4.1.7 Distributed Transactions and Spatial Index Consistency

In a distributed transaction, different branches of the transaction can execute in different sessions. The branches can detach from their current session and migrate to another within the transaction scope. To maintain the consistency of Spatial indexes in distributed transactions, you must follow the usage guidelines in this section.

When the first insert, update, or delete operation on a spatial table (one with a spatial index) is performed in a distributed transaction, all subsequent insert, update, or delete operations on the table, as well as any prepare to commit operation (the first branch to prepare a commit), in the transaction should happen *in the same session* as the first operation. The branches performing these subsequent operations will first have to connect to the session in which the first operation was performed.

For more information about distributed transactions, see *Oracle Database Administrator's Guide*.

4.2 Querying Spatial Data

This section describes how the structures of a Spatial layer are used to resolve spatial queries and spatial joins.

Spatial uses a two-tier query model with primary and secondary filter operations to resolve spatial queries and spatial joins, as explained in [Section 1.6](#). The term *two-tier* indicates that two distinct operations are performed to resolve queries. If both operations are performed, the exact result set is returned.

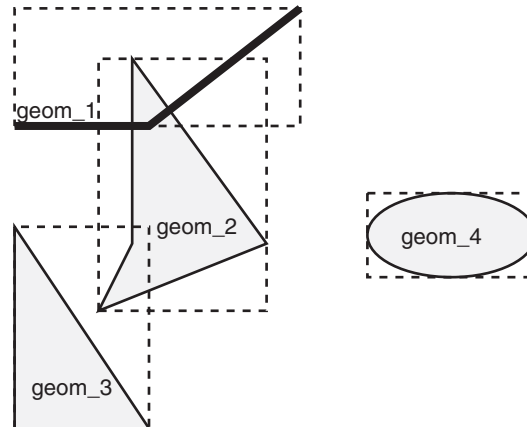
You cannot append a database link (dblink) name to the name of a spatial table in a query if a spatial index is defined on that table.

If a spatial index is created in a database that was created using the UTF8 character set, spatial queries that use the spatial index will fail if the system parameter NLS_LENGTH_SEMANTICS is set to CHAR. For spatial queries to succeed in this case, the NLS_LENGTH_SEMANTICS parameter must be set to BYTE (its default value).

4.2.1 Spatial Query

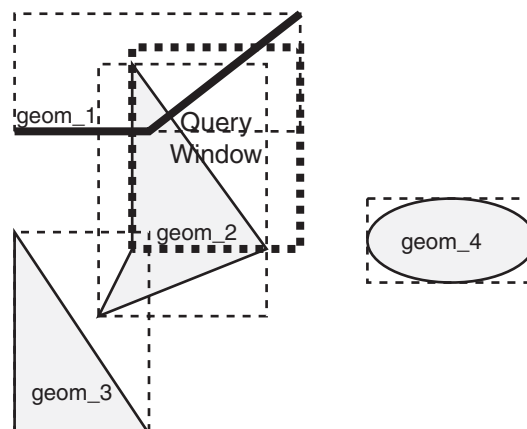
In a spatial R-tree index, each geometry is represented by its minimum bounding rectangle (MBR), as explained in [Section 1.7.1](#). Consider the following layer containing several objects in [Figure 4-1](#). Each object is labeled with its geometry name (geom_1 for the line string, geom_2 for the four-sided polygon, geom_3 for the triangular polygon, and geom_4 for the ellipse), and the MBR around each object is represented by a dashed line.

Figure 4-1 Geometries with MBRs



A typical spatial query is to request all objects that lie within a **query window**, that is, a defined fence or window. A dynamic query window refers to a rectangular area that is not defined in the database, but that must be defined before it is used. [Figure 4-2](#) shows the same geometries as in [Figure 4-1](#), but adds a query window represented by the heavy dotted-line box.

Figure 4-2 Layer with a Query Window



In [Figure 4-2](#), the query window covers parts of geometries geom_1 and geom_2, as well as part of the MBR for geom_3 but none of the actual geom_3 geometry. The query window does not cover any part of the geom_4 geometry or its MBR.

4.2.1.1 Primary Filter Operator

The `SDO_FILTER` operator, described in [Chapter 11](#), implements the primary filter portion of the two-step process involved in the Oracle Spatial query processing model. The primary filter uses the index data to determine only if a set of candidate object pairs may interact. Specifically, the primary filter checks to see if the MBRs of the candidate objects interact, not whether the objects themselves interact. The `SDO_FILTER` operator syntax is as follows:

```
SDO_FILTER(geometry1 SDO_GEOMETRY, geometry2 SDO_GEOMETRY, param VARCHAR2)
```

In the preceding syntax:

- `geometry1` is a column of type `SDO_GEOMETRY` in a table. This column must be spatially indexed.
- `geometry2` is an object of type `SDO_GEOMETRY`. This object may or may not come from a table. If it comes from a table, it may or may not be spatially indexed.
- `param` is an optional string of type `VARCHAR2`. It can specify either or both of the `min_resolution` and `max_resolution` keywords.

The following examples perform a primary filter operation only (with no secondary filter operation). They will return all the geometries shown in [Figure 4-2](#) that have an MBR that interacts with the query window. The result of the following examples are geometries `geom_1`, `geom_2`, and `geom_3`.

[Example 4-1](#) performs a primary filter operation without inserting the query window into a table. The window will be indexed in memory and performance will be very good.

Example 4-1 Primary Filter with a Temporary Query Window

```
SELECT A.Feature_ID FROM TARGET A
WHERE sdo_filter(A.shape, SDO_geometry(2003,NULL,NULL,
                                SDO_elem_info_array(1,1003,3),
                                SDO_ordinate_array(x1,y1, x2,y2))
) = 'TRUE';
```

In [Example 4-1](#), $(x1, y1)$ and $(x2, y2)$ are the lower-left and upper-right corners of the query window.

In [Example 4-2](#), a transient instance of type `SDO_GEOMETRY` was constructed for the query window instead of specifying the window parameters in the query itself.

Example 4-2 Primary Filter with a Transient Instance of the Query Window

```
SELECT A.Feature_ID FROM TARGET A
WHERE sdo_filter(A.shape, :theWindow) = 'TRUE';
```

[Example 4-3](#) assumes the query window was inserted into a table called `WINDOWS`, with an ID of `WINS_1`.

Example 4-3 Primary Filter with a Stored Query Window

```
SELECT A.Feature_ID FROM TARGET A, WINDOWS B
WHERE B.ID = 'WINS_1' AND
      sdo_filter(A.shape, B.shape) = 'TRUE';
```

If the `B.SHAPE` column is not spatially indexed, the `SDO_FILTER` operator indexes the query window in memory and performance is very good.

4.2.1.2 Primary and Secondary Filter Operator

The `SDO_RELATE` operator, described in [Chapter 11](#), performs both the primary and secondary filter stages when processing a query. The secondary filter ensures that only candidate objects that actually interact are selected. This operator can be used only if a spatial index has been created on two dimensions of data. The syntax of the `SDO_RELATE` operator is as follows:

```
SDO_RELATE(geometry1 SDO_GEOMETRY,
            geometry2 SDO_GEOMETRY,
            param      VARCHAR2)
```

In the preceding syntax:

- `geometry1` is a column of type `SDO_GEOMETRY` in a table. This column must be spatially indexed.
- `geometry2` is an object of type `SDO_GEOMETRY`. This object may or may not come from a table. If it comes from a table, it may or may not be spatially indexed.
- `param` is a quoted string with the mask keyword and a valid mask value, and optionally either or both of the `min_resolution` and `max_resolution` keywords, as explained in the documentation for the `SDO_RELATE` operator in [Chapter 11](#).

The following examples perform both primary and secondary filter operations. They return all the geometries in [Figure 4-2](#) that lie within or overlap the query window. The result of these examples is objects `geom_1` and `geom_2`.

[Example 4-4](#) performs both primary and secondary filter operations without inserting the query window into a table. The window will be indexed in memory and performance will be very good.

Example 4-4 Secondary Filter Using a Temporary Query Window

```
SELECT A.Feature_ID FROM TARGET A
       WHERE sdo_relate(A.shape, SDO_geometry(2003,NULL,NULL,
                                             SDO_elem_info_array(1,1003,3),
                                             SDO_ordinate_array(x1,y1, x2,y2)),
                       'mask=anyinteract') = 'TRUE';
```

In [Example 4-4](#), $(x1, y1)$ and $(x2, y2)$ are the lower-left and upper-right corners of the query window.

[Example 4-5](#) assumes the query window was inserted into a table called `WINDOWS`, with an ID value of `WINS_1`.

Example 4-5 Secondary Filter Using a Stored Query Window

```
SELECT A.Feature_ID FROM TARGET A, WINDOWS B
       WHERE B.ID = 'WINS_1' AND
             sdo_relate(A.shape, B.shape,
                       'mask=anyinteract') = 'TRUE';
```

If the `B.SHAPE` column is not spatially indexed, the `SDO_RELATE` operator indexes the query window in memory and performance is very good.

4.2.1.3 Within-Distance Operator

The `SDO_WITHIN_DISTANCE` operator, described in [Chapter 11](#), is used to determine the set of objects in a table that are within n distance units from a reference object. This operator can be used only if a spatial index has been created on two

dimensions of data. The reference object may be a transient or persistent instance of SDO_GEOMETRY (such as a temporary query window or a permanent geometry stored in the database). The syntax of the operator is as follows:

```
SDO_WITHIN_DISTANCE(geometry1 SDO_GEOMETRY,
                    aGeom      SDO_GEOMETRY,
                    params     VARCHAR2);
```

In the preceding syntax:

- geometry1 is a column of type SDO_GEOMETRY in a table. This column must be spatially indexed.
- aGeom is an instance of type SDO_GEOMETRY.
- params is a quoted string of keyword value pairs that determines the behavior of the operator. See the [SDO_WITHIN_DISTANCE](#) operator in [Chapter 11](#) for a list of parameters.

The following example selects any objects within 1.35 distance units from the query window:

```
SELECT A.Feature_ID
FROM TARGET A
WHERE SDO_WITHIN_DISTANCE( A.shape, :theWindow, 'distance=1.35') = 'TRUE';
```

The distance units are based on the geometry coordinate system in use. If you are using a geodetic coordinate system, the units are meters. If no coordinate system is used, the units are the same as for the stored data.

The [SDO_WITHIN_DISTANCE](#) operator is not suitable for performing spatial joins. That is, a query such as *Find all parks that are within 10 distance units from coastlines* will not be processed as an index-based spatial join of the COASTLINES and PARKS tables. Instead, it will be processed as a nested loop query in which each COASTLINES instance is in turn a reference object that is buffered, indexed, and evaluated against the PARKS table. Thus, the [SDO_WITHIN_DISTANCE](#) operation is performed *n* times if there are *n* rows in the COASTLINES table.

For non-geodetic data, there is an efficient way to accomplish a spatial join that involves buffering all geometries of a layer. This method does not use the [SDO_WITHIN_DISTANCE](#) operator. First, create a new table COSINE_BUFS as follows:

```
CREATE TABLE cosine_bufs UNRECOVERABLE AS
SELECT SDO_BUFFER (A.SHAPE, B.DIMINFO, 1.35)
FROM COSINE A, USER_SDO_GEOM_METADATA B
WHERE TABLE_NAME='COSINES' AND COLUMN_NAME='SHAPE';
```

Next, create a spatial index on the SHAPE column of COSINE_BUFS. Then you can perform the following query:

```
SELECT /*+ ordered */ a.gid, b.gid
FROM TABLE(SDO_JOIN('PARKS', 'SHAPE',
                    'COSINE_BUFS', 'SHAPE',
                    'mask=ANYINTERACT')) c,
parks a,
cosine_bufs b
WHERE c.rowid1 = a.rowid AND c.rowid2 = b.rowid;
```


4.2.1.4 Nearest Neighbor Operator

The `SDO_NN` operator, described in [Chapter 11](#), is used to identify the nearest neighbors for a geometry. This operator can be used only if a spatial index has been created on two dimensions of data. The syntax of the operator is as follows:

```
SDO_NN(geometry1 SDO_GEOMETRY,
        geometry2 SDO_GEOMETRY,
        param     VARCHAR2
        [, number NUMBER]);
```

In the preceding syntax:

- `geometry1` is a column of type `SDO_GEOMETRY` in a table. This column must be spatially indexed.
- `geometry2` is an instance of type `SDO_GEOMETRY`.
- `param` is a quoted string of keyword-value pairs that can determine the behavior of the operator, such as how many nearest neighbor geometries are returned. See the `SDO_NN` operator in [Chapter 11](#) for information about this parameter.
- `number` is the same number used in the call to `SDO_NN_DISTANCE`. Use this only if the `SDO_NN_DISTANCE` ancillary operator is included in the call to `SDO_NN`. See the `SDO_NN` operator in [Chapter 11](#) for information about this parameter.

The following example finds the two objects from the `SHAPE` column in the `COLA_MARKETS` table that are closest to a specified point (10,7). (Note the use of the optimizer hint in the `SELECT` statement, as explained in the Usage Notes for the `SDO_NN` operator in [Chapter 11](#).)

```
SELECT /*+ INDEX(cola_markets cola_spatial_idx) */
       c.mkt_id, c.name FROM cola_markets c WHERE SDO_NN(c.shape,
       SDO_geometry(2001, NULL, SDO_point_type(10,7,NULL), NULL,
       NULL), 'sdo_num_res=2') = 'TRUE';
```

4.2.1.5 Spatial Functions

Spatial also supplies functions for determining relationships between geometries, finding information about single geometries, changing geometries, and combining geometries. These functions all take into account two dimensions of source data. If the output value of these functions is a geometry, the resulting geometry will have the same dimensionality as the input geometry, but only the first two dimensions will accurately reflect the result of the operation.

4.2.2 Spatial Join

A **spatial join** is the same as a regular join except that the predicate involves a spatial operator. In Spatial, a spatial join takes place when you compare all geometries of one layer to all geometries of another layer. This is unlike a query window, which compares a single geometry to all geometries of a layer.

Spatial joins can be used to answer questions such as *Which highways cross national parks?*

The following table structures illustrate how the join would be accomplished for this example:

```
PARKS(   GID VARCHAR2(32), SHAPE SDO_GEOMETRY)
HIGHWAYS( GID VARCHAR2(32), SHAPE SDO_GEOMETRY)
```

To perform a spatial join, use the `SDO_JOIN` operator, which is described in [Chapter 11](#). The following spatial join query, to list the GID column values of highways and parks where a highway intersects with a park, performs a primary filter operation only ('mask=FILTER'), and thus it returns only approximate results:

```
SELECT /*+ ordered */ a.gid, b.gid
  FROM TABLE(SDO_JOIN('PARKS', 'SHAPE',
                       'HIGHWAYS', 'SHAPE',
                       'mask=FILTER')) c,
        parks a,
        highways b
 WHERE c.rowid1 = a.rowid AND c.rowid2 = b.rowid;
```

The following spatial join query requests the same information as in the preceding example, but it performs both primary and secondary filter operations ('mask=ANYINTERACT'), and thus it returns exact results:

```
SELECT /*+ ordered */ a.gid, b.gid
  FROM TABLE(SDO_JOIN('PARKS', 'SHAPE',
                       'HIGHWAYS', 'SHAPE',
                       'mask=ANYINTERACT')) c,
        parks a,
        highways b
 WHERE c.rowid1 = a.rowid AND c.rowid2 = b.rowid;
```

4.2.3 Cross-Schema Operator Invocation

You can invoke spatial operators on an indexed table that is not in your schema. Assume that user A has a spatial table T1 (with index table `IDX_TAB1`) with a spatial index defined, that user B has a spatial table T2 (with index table `IDX_TAB2`) with a spatial index defined, and that user C wants to invoke operators on tables in one or both of the other schemas.

If user C wants to invoke an operator only on T1, user C must perform the following steps:

1. Connect as user A and execute the following statements:

```
GRANT select on T1 to C;
GRANT select on idx_tab1 to C;
```

2. Connect as user C and execute a statement such as the following:

```
SELECT a.gid
  FROM T1 a
 WHERE sdo_filter(a.geometry, :theGeometry) = 'TRUE';
```

If user C wants to invoke an operator on both T1 and T2, user C must perform the following steps:

1. Connect as user A and execute the following statements:

```
GRANT select on T1 to C;
GRANT select on idx_tab1 to C;
```

2. Connect as user B and execute the following statements:

```
GRANT select on T2 to C;
GRANT select on idx_tab2 to C;
```

3. Connect as user C and execute a statement such as the following:

```
SELECT a.gid
```

```
FROM T1 a, T2 b
WHERE b.gid = 5 AND
      sdo_filter(a.geometry, b.geometry) = 'TRUE';
```

Geocoding Address Data

Geocoding is the process of associating spatial locations (longitude and latitude coordinates) with postal addresses. This chapter includes the following major sections:

- [Section 5.1, "Concepts for Geocoding"](#)
- [Section 5.2, "Data Types for Geocoding"](#)
- [Section 5.3, "Using the Geocoding Capabilities"](#)
- [Section 5.4, "Geocoding from a Place Name"](#)
- [Section 5.5, "Data Structures for Geocoding"](#)

5.1 Concepts for Geocoding

This section describes concepts that you must understand before you use the Spatial geocoding capabilities.

5.1.1 Address Representation

Addresses to be geocoded can be represented either as formatted addresses or unformatted addresses.

A **formatted address** is described by a set of attributes for various parts of the address, which can include some or all of those shown in [Table 5–1](#).

Table 5–1 *Attributes for Formal Address Representation*

Address Attribute	Description
Name	Place name (optional).
Intersecting street	Intersecting street name (optional).
Street	Street address, including the house or building number, street name, street type (Street, Road, Blvd, and so on), and possibly other information. In the current release, the first four characters of the street name must match a street name in the geocoding data for there to be a potential street name match.
Settlement	The lowest-level administrative area to which the address belongs. In most cases it is the city. In some European countries, the settlement can be an area within a large city, in which case the large city is the municipality.
Municipality	The administrative area above settlement. Municipality is not used for United States addresses. In European countries where cities contain settlements, the municipality is the city.

Table 5–1 (Cont.) Attributes for Formal Address Representation

Address Attribute	Description
Region	The administrative area above municipality (if applicable), or above settlement if municipality does not apply. In the United States, the region is the state; in some other countries, the region is the province.
Postal code	Postal code (optional if administrative area information is provided). In the United States, the postal code is the 5-digit ZIP code.
Postal add-on code	String appended to the postal code. In the United States, the postal add-on code is typically the last four numbers of a 9-digit ZIP code specified in "5-4" format.
Country	The country name or ISO country code.

Formatted addresses are specified using the SDO_GEO_ADDR data type, which is described in [Section 5.2.1](#).

An **unformatted address** is described using lines with information in the postal address format for the relevant country. The address lines must contain information essential for geocoding, and they might also contain information that is not needed for geocoding (something that is common in unprocessed postal addresses). An unformatted address is stored as an array of strings. For example, an address might consist of the following strings: '22 Monument Square' and 'Concord, MA 01742'.

Unformatted addresses are specified using the SDO_KEYWORDARRAY data type, which is described in [Section 5.2.3](#).

5.1.2 Match Modes

The match mode for a geocoding operation determines how closely the attributes of an input address must match the data being used for the geocoding. Input addresses can include different ways of representing the same thing (such as *Street* and the abbreviation *St*), and they can include minor errors (such as the wrong postal code, even though the street address and city are correct and the street address is unique within the city).

You can require an exact match between the input address and the data used for geocoding, or you can relax the requirements for some attributes so that geocoding can be performed despite certain discrepancies or errors in the input addresses. [Table 5–2](#) lists the match modes and their meanings. Use a value from this table with the `MatchMode` attribute of the SDO_GEO_ADDR data type (described in [Section 5.2.1](#)) and for the `match_mode` parameter of a geocoding function or procedure.

Table 5–2 Match Modes for Geocoding Operations

Match Mode	Description
EXACT	All attributes of the input address must match the data used for geocoding. However, if the house or building number, base name (street name), street type, street prefix, and street suffix do not all match the geocoding data, a location in the first match found in the following is returned: postal code, city or town (settlement) within the state, and state. For example, if the street name is incorrect but a valid postal code is specified, a location in the postal code is returned.

Table 5–2 (Cont.) Match Modes for Geocoding Operations

Match Mode	Description
RELAX_STREET_TYPE	The street type can be different from the data used for geocoding. For example, if <i>Main St</i> is in the data used for geocoding, <i>Main Street</i> would also match that, as would <i>Main Blvd</i> if there was no <i>Main Blvd</i> and no other street type named <i>Main</i> in the relevant area.
RELAX_POI_NAME	The name of the point of interest does not have to match the data used for geocoding. For example, if <i>Jones State Park</i> is in the data used for geocoding, <i>Jones State Pk</i> and <i>Jones Park</i> would also match as long as there were no ambiguities or other matches in the data.
RELAX_HOUSE_NUMBER	The house or building number and street type can be different from the data used for geocoding. For example, if <i>123 Main St</i> is in the data used for geocoding, <i>123 Main Lane</i> and <i>124 Main St</i> would also match as long as there were no ambiguities or other matches in the data.
RELAX_BASE_NAME	The base name of the street, the house or building number, and the street type can be different from the data used for geocoding. For example, if <i>Pleasant Valley</i> is the base name of a street in the data used for geocoding, <i>Pleasant Vale</i> would also match as long as there were no ambiguities or other matches in the data.
RELAX_POSTAL_CODE	The postal code (if provided), base name, house or building number, and street type can be different from the data used for geocoding.
RELAX_BUILTUP_AREA	The address can be outside the city specified as long as it is within the same county. Also includes the characteristics of RELAX_POSTAL_CODE.
RELAX_ALL	Equivalent to RELAX_BUILTUP_AREA.
DEFAULT	Equivalent to RELAX_BASE_NAME.

5.1.3 Match Codes

The match code is a number indicating which input address attributes matched the data used for geocoding. The match code is stored in the `MatchCode` attribute of the output `SDO_GEO_ADDR` object (described in [Section 5.2.1](#)).

[Table 5–3](#) lists the possible match code values.

Table 5–3 Match Codes for Geocoding Operations

Match Code	Description
1	Exact match: the city name, postal code, street base name, street type (and suffix or prefix or both, if applicable), and house or building number match the data used for geocoding.
2	The city name, postal code, street base name, and house or building number match the data used for geocoding, but the street type, suffix, or prefix does not match.
3	The city name, postal code, and street base name match the data used for geocoding, but the house or building number does not match.
4	The city name and postal code match the data used for geocoding, but the street address does not match.
10	The city name matches the data used for geocoding, but the postal code does not match.

Table 5–3 (Cont.) Match Codes for Geocoding Operations

Match Code	Description
11	The postal code matches the data used for geocoding, but the city name does not match.

5.1.4 Error Messages for Output Geocoded Addresses

For an output geocoded address, the `ErrorMessage` attribute of the `SDO_GEO_ADDR` object (described in [Section 5.2.1](#)) contains a string that indicates which address attributes have been matched against the data used for geocoding. Before the geocoding operation begins, the string is set to the value `????????????281C??`; and the value is modified to reflect which attributes have been matched.

[Table 5–4](#) lists the character positions in the string and the address attribute corresponding to each position. It also lists the character value that the position is set to if the attribute is matched.

Table 5–4 Geocoded Address Error Message Interpretation

Position	Attribute	Value If Matched
1-4	(Reserved for future use)	???
5	House or building number	#
6	Street prefix	E
7	Street base name	N
8	Street suffix	U
9	Street type	T
10	Secondary unit	S
11	Built-up area or city	B
14	Region	1
15	Country	C
16	Postal code	P
17	Postal add-on code	A

5.2 Data Types for Geocoding

This section describes the data types specific to geocoding functions and procedures.

5.2.1 SDO_GEO_ADDR Type

The `SDO_GEO_ADDR` object type is used to describe an address. When a geocoded address is output by an `SDO_GCDR` function or procedure, it is stored as an object of type `SDO_GEO_ADDR`.

[Table 5–5](#) lists the attributes of the `SDO_GEO_ADDR` type. Not all attributes will be relevant in any given case. The attributes used for a returned geocoded address depend on the geographical context of the input address, especially the country.

Table 5–5 SDO_GEO_ADDR Type Attributes

Attribute	Data Type	Description
Id	NUMBER	(Not used.)
AddressLines	SDO_KEYWORDARRAY	Address lines. (The SDO_KEYWORDARRAY type is described in Section 5.2.3 .)
PlaceName	VARCHAR2(200)	Point of interest (POI) name. Example: <i>CALIFORNIA PACIFIC MEDICAL CTR</i>
StreetName	VARCHAR2(200)	Street name, including street type. Example: <i>MAIN ST</i>
IntersectStreet	VARCHAR2(200)	Intersecting street.
SecUnit	VARCHAR2(200)	Secondary unit, such as an apartment number or building number.
Settlement	VARCHAR2(200)	Lowest-level administrative area to which the address belongs. (See Table 5–1 .)
Municipality	VARCHAR2(200)	Administrative area above settlement. (See Table 5–1 .)
Region	VARCHAR2(200)	Administrative area above municipality (if applicable), or above settlement if municipality does not apply. (See Table 5–1 .)
Country	VARCHAR2(100)	Country name or ISO country code.
PostalCode	VARCHAR2(20)	Postal code (optional if administrative area information is provided). In the United States, the postal code is the 5-digit ZIP code.
PostalAddOnCode	VARCHAR2(20)	String appended to the postal code. In the United States, the postal add-on code is typically the last four numbers of a 9-digit ZIP code specified in "5-4" format.
FullPostalCode	VARCHAR2(20)	Full postal code, including the postal code and postal add-on code.
POBox	VARCHAR2(100)	Post Office box number.
HouseNumber	VARCHAR2(100)	House or building number. Example: <i>123</i> in <i>123 MAIN ST</i>
BaseName	VARCHAR2(200)	Base name of the street. Example: <i>MAIN</i> in <i>123 MAIN ST</i>
StreetType	VARCHAR2(20)	Type of the street. Example: <i>ST</i> in <i>123 MAIN ST</i>
StreetTypeBefore	VARCHAR2(1)	(Not used.)
StreetTypeAttached	VARCHAR2(1)	(Not used.)
StreetPrefix	VARCHAR2(20)	Prefix for the street. Example: <i>S</i> in <i>123 S MAIN ST</i>
StreetSuffix	VARCHAR2(20)	Suffix for the street. Example: <i>NE</i> in <i>123 MAIN ST NE</i>
Side	VARCHAR2(1)	Side of the street (L for left or R for right) that the house is on when you are traveling along the road segment following its orientation (that is, from its start node toward its end node). The house numbers may be increasing or decreasing.

Table 5–5 (Cont.) SDO_GEO_ADDR Type Attributes

Attribute	Data Type	Description
Percent	NUMBER	Number from 0 to 1 (multiply by 100 to get a percentage value) indicating how far along the street you are when traveling following the road segment orientation.
EdgeID	NUMBER	Edge ID of the road segment.
ErrorMessage	VARCHAR2(20)	Error message (see Section 5.1.4).
MatchCode	NUMBER	Match code (see Section 5.1.3).
MatchMode	VARCHAR2(30)	Match mode (see Section 5.1.2).
Longitude	NUMBER	Longitude coordinate value.
Latitude	NUMBER	Latitude coordinate value.

You can return the entire SDO_GEO_ADDR object, or you can specify an attribute using standard "dot" notation. [Example 5–1](#) contains statements that geocode the address of the San Francisco City Hall; the first statement returns the entire SDO_GEO_ADDR object, and the remaining statements return some specific attributes.

Example 5–1 Geocoding, Returning Address Object and Specific Attributes

```

SELECT SDO_GCDR.GEOCODE('SCOTT',
  SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA 94102'),
  'US', 'RELAX_BASE_NAME') FROM DUAL;

SDO_GCDR.GEOCODE('CJMURRAY',SDO_KEYWORDARRAY('1CARLTONBGODLETTPL','SANFRANCISCO
-----
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), NULL, 'CARLTON B GOODLETT PL', NULL, NULL, '
SAN FRANCISCO', NULL, 'CA', 'US', '94102', NULL, '94102', NULL, '1', 'CARLTON B
GOODLETT', 'PL', 'F', 'F', NULL, NULL, 'L', .01, 23614360, 'nul?#ENUT?B281CP?',
1, 'DEFAULT', -122.41815, 37.7784183)

SELECT SDO_GCDR.GEOCODE('SCOTT',
  SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA 94102'),
  'US', 'RELAX_BASE_NAME').StreetType FROM DUAL;

SDO_GCDR.GEOCODE('SCOTT',SDO_KEYWORDARRAY('1CARLTONBGODLETTPL','SANFRANCISCO
-----
PL

SELECT SDO_GCDR.GEOCODE('SCOTT',
  SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA 94102'),
  'US', 'RELAX_BASE_NAME').Side FROM DUAL;

S
-
L

SELECT SDO_GCDR.GEOCODE('SCOTT',
  SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA 94102'),
  'US', 'RELAX_BASE_NAME').Percent FROM DUAL;

SDO_GCDR.GEOCODE('SCOTT',SDO_KEYWORDARRAY('1CARLTONBGODLETTPL','SANFRANCISCO
-----

```

.01

```

SELECT SDO_GCDR.GEOCODE('SCOTT',
  SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA 94102'),
  'US', 'RELAX_BASE_NAME').EdgeID FROM DUAL;

SDO_GCDR.GEOCODE('SCOTT',SDO_KEYWORDARRAY('1CARLTONBGOODLETTPL','SANFRANCISCO
-----
23614360

SELECT SDO_GCDR.GEOCODE('SCOTT',
  SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA 94102'),
  'US', 'RELAX_BASE_NAME').MatchCode FROM DUAL;

SDO_GCDR.GEOCODE('SCOTT',SDO_KEYWORDARRAY('1CARLTONBGOODLETTPL','SANFRANCISCO
-----
1

```

5.2.2 SDO_ADDR_ARRAY Type

The SDO_ADDR_ARRAY type is a VARRAY of SDO_GEO_ADDR objects (described in [Section 5.2.1](#)) used to store geocoded address results. Multiple address objects can be returned when multiple addresses are matched as a result of a geocoding operation.

The SDO_ADDR_ARRAY type is defined as follows:

```
CREATE TYPE sdo_addr_array AS VARRAY(1000) OF sdo_geo_addr;
```

5.2.3 SDO_KEYWORDARRAY Type

The SDO_KEYWORDARRAY type is a VARRAY of VARCHAR2 strings used to store address lines for unformatted addresses. (Formatted and unformatted addresses are described in [Section 5.1.1](#).)

The SDO_KEYWORDARRAY type is defined as follows:

```
CREATE TYPE sdo_keywordarray AS VARRAY(10000) OF VARCHAR2(9000);
```

5.3 Using the Geocoding Capabilities

To use the Oracle Spatial geocoding capabilities, you must use data provided by a geocoding vendor, and the data must be in the format supported by the Oracle Spatial geocoding feature. For information about getting and loading this data, go to the Spatial page of the Oracle Technology Network (OTN):

<http://www.oracle.com/technology/products/spatial/>

Find the link for geocoding, and follow the instructions.

To geocode an address using the geocoding data, use the SDO_GCDR PL/SQL package subprograms, which are documented in [Chapter 14](#):

- The [SDO_GCDR.GEOCODE](#) function geocodes an unformatted address to return an SDO_GEO_ADDR object.
- The [SDO_GCDR.GEOCODE_ADDR](#) function geocodes an input address using attributes in an SDO_GEO_ADDR object, and returns the first matched address as an SDO_GEO_ADDR object.
- The [SDO_GCDR.GEOCODE_ADDR_ALL](#) function geocodes an input address using attributes in an SDO_GEO_ADDR object, and returns matching addresses as an SDO_ADDR_ARRAY object.

- The [SDO_GCDR.GEOCODE_AS_GEOMETRY](#) function geocodes an unformatted address to return an SDO_GEOMETRY object.
- The [SDO_GCDR.GEOCODE_ALL](#) function geocodes all addresses associated with an unformatted address and returns the result as an SDO_ADDR_ARRAY object (an array of address objects).
- The [SDO_GCDR.REVERSE_GEOCODE](#) function reverse geocodes a location, specified by its spatial geometry object and country, and returns the result as an SDO_GEO_ADDR object.

5.4 Geocoding from a Place Name

If you know a place name (point of interest) but not its locality details, you can create a PL/SQL function to construct an SDO_GEO_ADDR object from `placename` and `country` input parameters, as shown in [Example 5-2](#), which creates a function named `create_addr_from_placename`. The SELECT statement in this example uses the [SDO_GCDR.GEOCODE_ADDR](#) function to geocode the address constructed using the `create_addr_from_placename` function.

Example 5-2 Geocoding from a Place Name and Country

```
create or replace function create_addr_from_placename(
placename varchar2,
country varchar2)
return sdo_geo_addr
as
  addr sdo_geo_addr ;
begin
  addr := sdo_geo_addr() ;
  addr.country := country ;
  addr.placename := placename ;
  addr.matchmode := 'default' ;
  return addr ;
end;
/

SELECT sdo_gcdr.geocode_addr('SCOTT',
  create_addr_from_placename('CALIFORNIA PACIFIC MEDICAL CTR', 'US'))
FROM DUAL;
```

If you know at least some of the locality information, such as settlement, region, and postal code, you can get better performance if you can provide such information. [Example 5-3](#) provides an alternate version of the `create_addr_from_placename` function that accepts additional parameters. To call this version of the function, specify actual values for the `placename` and `country` parameters, and specify an actual value or a null value for each of the other input parameters.

Example 5-3 Geocoding from a Place Name, Country, and Other Fields

```
create or replace function create_addr_from_placename(
placename varchar2,
city varchar2,
state varchar2,
postalcode varchar2,
country varchar2)
return sdo_geo_addr
as
  addr sdo_geo_addr ;
```

```

begin
  addr := sdo_geo_addr() ;
  addr.settlement := city ;
  addr.region := state ;
  addr.postalcode := postalcode ;
  addr.country := country ;
  addr.placename := placename ;
  addr.matchmode := 'default' ;
  return addr ;
end;
/

SELECT sdo_gcdr.geocode_addr('SCOTT',
  create_addr_from_placename('CALIFORNIA PACIFIC MEDICAL CTR',
    'san francisco', 'ca', null, 'US')) FROM DUAL;

```

5.5 Data Structures for Geocoding

Oracle uses the following tables for geocoding:

- GC_PARSER_PROFILES
- GC_PARSER_PROFILEAFS
- GC_COUNTRY_PROFILE
- GC_AREA_<suffix>
- GC_POSTAL_CODE_<suffix>
- GC_ROAD_SEGMENT_<suffix>
- GC_ROAD_<suffix>
- GC_POI_<suffix>
- GC_INTERSECTION_<suffix>

The GC_PARSER_PROFILES and GC_PARSER_PROFILEAFS tables store address format definitions of all supported counties. These tables are used by the internal address parser in parsing postal addresses into addressing fields. The data for these two tables is provided by Oracle. The remaining tables store geocoding data provided by data vendors.

Each user that owns the tables containing geocoding data (that is, each user that can be specified with the `username` parameter in a call to an SDO_GCDR subprogram) must have one GC_PARSER_PROFILES table, one GC_PARSER_PROFILEAFS table, and one GC_COUNTRY_PROFILE table. Each such user can have multiple sets of the other tables (GC_XXX_<suffix>). Each set of tables whose names end with the same suffix stores geocoding data of a country. For example, the following set of tables can be used to store geocoding data of the United States:

- GC_AREA_US
- GC_POSTAL_CODE_US
- GC_ROAD_SEGMENT_US
- GC_ROAD_US
- GC_POI_US
- GC_INTERSECTION_US

Geocoding data of one country cannot be stored in more than one set of those tables. The table suffix is defined by data vendors and is specified in the GC_TABLE_SUFFIX column in the GC_COUNTRY_PROFILE table (described in [Section 5.5.2](#)).

The following sections describe the vendor-supplied tables that store geocoding data, in alphabetical order by table name.

5.5.1 GC_AREA_<suffix> Table

The GC_AREA_<suffix> table (for example, CG_AREA_US) stores administration area information for the country associated with the table name suffix. This table contains one row for each administration area, and it contains the columns shown in [Table 5–6](#).

Table 5–6 GC_AREA_<suffix> Table

Column Name	Data Type	Description
AREA_ID	NUMBER(10)	Area ID number. (Required)
AREA_NAME	VARCHAR2(64)	Area name. (Required)
LANG_CODE	VARCHAR2(3)	3-letter ISO national language code for the language associated with the area. (Required)
ADMIN_LEVEL	NUMBER(1)	Administration hierarchy level for the area. (Required)
LEVEL1_AREA_ID	NUMBER(10)	ID of the level-1 area to which the area belongs. In the administration hierarchy, the level-1 area is the country. (Required)
LEVEL2_AREA_ID	NUMBER(10)	ID of the level-2 area to which the area belongs, if applicable. You must specify an area ID for each level in the administration hierarchy to which this area belongs. (Optional)
LEVEL3_AREA_ID	NUMBER(10)	ID of the level-3 area to which the area belongs, if applicable. You must specify an area ID for each level in the administration hierarchy to which this area belongs. (Optional)
LEVEL4_AREA_ID	NUMBER(10)	ID of the level-4 area to which the area belongs, if applicable. You must specify an area ID for each level in the administration hierarchy to which this area belongs. (Optional)
LEVEL5_AREA_ID	NUMBER(10)	ID of the level-5 area to which the area belongs, if applicable. You must specify an area ID for each level in the administration hierarchy to which this area belongs. (Optional)
LEVEL6_AREA_ID	NUMBER(10)	ID of the level-6 area to which the area belongs, if applicable. You must specify an area ID for each level in the administration hierarchy to which this area belongs. (Optional)
LEVEL7_AREA_ID	NUMBER(10)	ID of the level-7 area to which the area belongs, if applicable. You must specify an area ID for each level in the administration hierarchy to which this area belongs. (Optional)
CENTER_LONG	NUMBER	Longitude value of the center of the area. The center is set to the closest road segment to the center longitude and latitude values. Oracle recommends that these two attributes be set properly. If these values are not set, the longitude and latitude coordinates of the geocoded result of an area will be (0,0). (Optional)

Table 5–6 (Cont.) GC_AREA_<suffix> Table

Column Name	Data Type	Description
CENTER_LAT	NUMBER	Latitude value of the center of the area. (See the explanation for the CENTER_LONG column.) (Optional)
ROAD_SEGMENT_ID	NUMBER(10)	ID of the road segment to which the area center is set. This value must be set correctly if the geocoder is intended to work with the Oracle Spatial routing engine (described in Appendix C); otherwise, it can be set to any nonzero value, but it cannot be null. (Required)
POSTAL_CODE	VARCHAR2(16)	Postal code for the center of the area. Oracle recommends that this attribute be set correctly. If this value is null, the postal code attribute of the geocoded result of an area will be null. (Optional)
COUNTRY_CODE_2	VARCHAR2(2)	2- letter ISO country code of the country to which the area belongs. (Required)
PARTITION_ID	NUMBER	Partition key used for partitioning geocoder data by geographic boundaries. If the data is not partitioned, set the value to 1. (Required)
REAL_NAME	VARCHAR2(64)	The real name of the area, as spelled using the local language. This column is useful for area names that are not in English. For example, the German name of city MUNICH is MÜNCHEN. It is allowed to be spelled as MÜNCHEN, but its REAL_NAME value should be MÜNCHEN. In the area table for Germany, areas with name MÜNCHEN and MUNCHEN both refer to the same area, and they both have the same real name MÜNCHEN. If the area name does not have any non-English characters, set REAL_NAME to be the same as AREA_NAME. (Required)
IS_ALIAS	VARCHAR2(1)	Contains T if this area is an alias of another area that is an officially recognized administrative area; contains F if this area is not an alias of another area that is an officially recognized administrative area. For example, Manhattan is not an officially recognized administrative area, but it is used by the public to refer to a part of New York City. In this case, Manhattan is an alias of New York City. (Required)
NUM_STREETS	NUMBER	The number of streets inside this area. (Optional)

5.5.2 GC_COUNTRY_PROFILE Table

The GC_COUNTRY_PROFILE table stores country profile information used by the geocoder. This table contains one row for each supported country, and it contains the columns shown in [Table 5–7](#).

Table 5–7 GC_COUNTRY_PROFILE Table

Column Name	Data Type	Description
COUNTRY_NAME	VARCHAR2(60)	Country name. (Required)
COUNTRY_CODE_3	VARCHAR2(3)	3- letter ISO country code. (Required)
COUNTRY_CODE_2	VARCHAR2(2)	2- letter ISO country code. (Required)

Table 5-7 (Cont.) GC_COUNTRY_PROFILE Table

Column Name	Data Type	Description
LANG_CODE_1	VARCHAR2(3)	3-letter ISO national language code. Some country might have multiple national languages, in which case LANG_CODE_2 and perhaps other columns should contain values. (Required)
LANG_CODE_2	VARCHAR2(3)	3-letter ISO national language code. (Optional)
LANG_CODE_3	VARCHAR2(3)	3-letter ISO national language code. (Optional)
LANG_CODE_4	VARCHAR2(3)	3-letter ISO national language code. (Optional)
NUMBER_ADMIN_LEVELS	NUMBER(1)	Number of administration hierarchy levels. A country can have up to 7 administration area levels, numbered from 1 to 7. The top level area (country) is level 1. For the United States, the administration hierarchy is as follows: level 1 = country, level 2 = state, level 3 = county, level 4 = city. (Required)
SETTLEMENT_LEVEL	NUMBER(1)	Administration hierarchy level for a settlement, which is the lowest area level used in addressing. In the United States, this is the city level. (Required)
MUNICIPALITY_LEVEL	NUMBER(1)	Administration hierarchy level for a municipality, which is the second-lowest area level used in addressing. In the United States, this is the county level. (Optional)
REGION_LEVEL	NUMBER(1)	Administration hierarchy level for the region level used in addressing. (Optional)
SETTLEMENT_IS_OPTIONAL	VARCHAR2(1)	Contains T if settlement information is required in the address data; contains F if settlement information is not required in the address data. (Required)
MUNICIPALITY_IS_OPTIONAL	VARCHAR2(1)	Contains T if municipality information is required in the address data; contains F if municipality information is not required in the address data. (Required)
REGION_IS_OPTIONAL	VARCHAR2(1)	Contains T if region information is required in the address data; contains F if region information is not required in the address data. (Required)
POSTCODE_IN_SETTLEMENT	VARCHAR(1)	Contains T if each postal code must be completely within a settlement area; contains F if a postal code can include area from multiple settlements. (Required)
SETTLEMENT_AS_CITY	VARCHAR(1)	Contains T if a city name can identify both a municipality and a settlement; contains F if a city name can identify only a settlement. For example, in the United Kingdom, London can be both the name of a municipality area and the name of a settlement area, which is inside the municipality of London. This is common in large cities in some European countries, such as the UK and Belgium. (Required)
CACHED_ADMIN_AREA_LEVEL	NUMBER	(Reserved for future use.)
GC_TABLE_SUFFIX	VARCHAR2(5)	Table name suffix identifying the country. For example, if the value of GC_TABLE_SUFFIX is US, the names of tables with geocoding data for this country end with _US (for example, CG_AREA_US). (Required)
CENTER_LONG	NUMBER	Longitude value of the center of the area. (Optional)
CENTER_LAT	NUMBER	Latitude value of the center of the area. (Optional)

Table 5–7 (Cont.) GC_COUNTRY_PROFILE Table

Column Name	Data Type	Description
SEPARATE_PREFIX	VARCHAR2(1)	Contains T if the street name prefix is a separate word from the street name; contains F if the street name prefix is in the same word with the street name. For example, in an American street address of 123 N Main St, the prefix is N, and it is separate from the street name, which is Main. (Optional; not currently used by Oracle)
SEPARATE_SUFFIX	VARCHAR2(1)	Contains T if the street name suffix is a separate word from the street name; contains F if the street name suffix is in the same word with the street name. For example, in an American street address of 123 Main St NW, the suffix is NW, and it is separate from the street name, which is Main, and from the street type, which is St. (Optional; not currently used by Oracle)
SEPARATE_STYPE	VARCHAR2(1)	Contains T if the street type is a separate word from the street name; contains F if the street type is in the same word with the street name. For example, in a German street address of 123 Beethovenstrass, the type is strass, and it is in the same word with the street name, which is Beethoven. (Optional; not currently used by Oracle)
AREA_ID	NUMBER	Not currently used by Oracle. (Optional)
VERSION	VARCHAR2(10)	Version of the data. The first version should be 1.0. (Required)

5.5.3 GC_INTERSECTION_<suffix> Table

The GC_INTERSECTION_<suffix> table (for example, GC_INTERSECTION_US) stores road intersection information. An intersection is typically associated with multiple roads. Each row represents an intersection and two different roads that intersect with each other at this intersection. This table contains the columns shown in [Table 5–8](#).

Table 5–8 GC_INTERSECTION_<suffix> Table

Column Name	Data Type	Description
ROAD_ID_1	NUMBER	ID number of the first road on which the intersection is located. (Required)
ROAD_SEGMENT_ID_1	NUMBER	ID number of the road segment on the first road on which the intersection is located. (Required)
ROAD_ID_2	NUMBER	ID number of the second road on which the intersection is located. (Required)
ROAD_SEGMENT_ID_2	NUMBER	ID number of the road segment on the second road on which the intersection is located. (Required)
INTS_LONG	NUMBER	Longitude coordinate value of the intersection. (Required)
INTS_LAT	NUMBER	Latitude coordinate value of the intersection. (Required)
HOUSE_NUMBER	NUMBER	The leading numerical part of the house number at the intersection. (See the explanation of house numbers after Table 5–12 in Section 5.5.7 .) (Required)

Table 5–8 (Cont.) GC_INTERSECTION_<suffix> Table

Column Name	Data Type	Description
HOUSE_NUMBER_2	VARCHAR2(10)	The second part of the house number at the intersection. (See the explanation of house numbers after Table 5–12 in Section 5.5.7 .) (Required)
SIDE	VARCHAR2(1)	Side of the street on which the house at the intersection is located. Possible values: L (left) or R (right). (Required)
COUNTRY_CODE_2	VARCHAR2(2)	2-letter ISO country code of the country to which the house at the intersection belongs. (Required)
PARTITION_ID	NUMBER	Partition key used for partitioning geocoder data by geographic boundaries. If the data is not partitioned, set the value to 1. (Required)

5.5.4 GC_POI_<suffix> Table

The GC_POI_<suffix> table (for example, GC_POI_US) stores point of interest (POI) information for the country associated with the table name suffix. This table contains one or more rows for each point of interest. (For example, it can contain multiple rows for a POI if the POI is associated with multiple settlements.) The GC_POI_<suffix> table contains the columns shown in [Table 5–9](#).

Table 5–9 GC_POI_<suffix> Table

Column Name	Data Type	Description
POI_ID	NUMBER	ID number of the POI. (Required)
POI_NAME	VARCHAR2(64)	Name of the POI. (Required)
LANG_CODE	VARCHAR2(3)	3-letter ISO national language code for the language for the POI name. (Required)
FEATURE_CODE	NUMBER	Feature code for the POI, if the data vendor classifies POIs by category. (Optional)
HOUSE_NUMBER	VARCHAR2(10)	House number of the POI; may contain non-numeric characters. (Required)
STREET_NAME	VARCHAR2(80)	Street name of the POI. (Required)
SETTLEMENT_ID	NUMBER(10)	ID number of the settlement to which the POI belongs. (Required if the POI is associated with a settlement)
MUNICIPALITY_ID	NUMBER(10)	ID number of the municipality to which the POI belongs. (Required if the POI is associated with a municipality)
REGION_ID	NUMBER(10)	ID number of the region to which the POI belongs. (Required if the POI is associated with a region)
SETTLEMENT_NAME	VARCHAR2(64)	Name of the settlement to which the POI belongs. (Required if the POI is associated with a settlement)
MUNICIPALITY_NAME	VARCHAR2(64)	Name of the municipality to which the POI belongs. (Required if the POI is associated with a municipality)
REGION_NAME	VARCHAR2(64)	Name of the region to which the postal code belongs. (Required if the POI is associated with a region)
POSTAL_CODE	VARCHAR2(16)	Name of the postal code of the POI. (Required)

Table 5–9 (Cont.) GC_POI_<suffix> Table

Column Name	Data Type	Description
VANITY_CITY	VARCHAR2(35)	Name of the city popularly associated with the POI, if it is different from the actual city containing the POI. For example, the London Heathrow Airport is actually located in a town named Hayes, which is part of greater London, but people tend to associate the airport only with London. In this case, the VANITY_CITY value is London. (Optional)
ROAD_SEGMENT_ID	NUMBER	ID of the road segment on which the POI is located. (Required)
SIDE	VARCHAR2(1)	Side of the street on which the POI is located. Possible values: L (left) or R (right). (Required)
PERCENT	NUMBER	Percentage value at which POI is located on the road. It is computed by dividing the distance from the street segment start point to the POI by the length of the street segment. (Required)
TELEPHONE_NUMBER	VARCHAR2(20)	Telephone number of the POI. (Optional)
LOC_LONG	NUMBER	Longitude coordinate value of the POI. (Required)
LOC_LAT	NUMBER	Latitude coordinate value of the POI. (Required)
COUNTRY_CODE_2	VARCHAR2(2)	2- letter ISO country code of the country to which the POI belongs. (Required)
PARTITION_ID	NUMBER	Partition key used for partitioning geocoder data by geographic boundaries. If the data is not partitioned, set the value to 1. (Required)

5.5.5 GC_POSTAL_CODE_<suffix> Table

The GC_POSTAL_CODE_<suffix> table (for example, GC_POSTAL_CODE_US) stores postal code information for the country associated with the table name suffix. This table contains one or more rows for each postal code. (For example, it can contain multiple rows for a postal code if the postal code is associated with multiple settlements.) The GC_POSTAL_CODE_<suffix> table contains the columns shown in [Table 5–10](#).

Table 5–10 GC_POSTAL_CODE_<suffix> Table

Column Name	Data Type	Description
POSTAL_CODE	VARCHAR2(16)	Postal code. (Required)
SETTLEMENT_NAME	VARCHAR2(64)	Name of the settlement to which the postal code belongs. (Required if the postal code is associated with a settlement)
MUNICIPALITY_NAME	VARCHAR2(64)	Name of the municipality to which the postal code belongs. (Required if the postal code is associated with a municipality)
REGION_NAME	VARCHAR2(64)	Name of the region to which the postal code belongs. (Required if the postal code is associated with a region)
LANG_CODE	VARCHAR2(3)	3-letter ISO national language code for the language associated with the area. (Required)
SETTLEMENT_ID	NUMBER(10)	ID number of the settlement to which the postal code belongs. (Required if the postal code is associated with a settlement)

Table 5–10 (Cont.) GC_POSTAL_CODE_<suffix> Table

Column Name	Data Type	Description
MUNICIPALITY_ID	NUMBER(10)	ID number of the municipality to which the postal code belongs. (Required if the postal code is associated with a municipality)
REGION_ID	NUMBER(10)	ID number of the region to which the postal code belongs. (Required if the postal code is associated with a region)
CENTER_LONG	NUMBER	Longitude value of the center of the area. The center is set to the closest road segment to the center longitude and latitude values. Oracle recommends that these two attributes be set properly. If these values are not set, the longitude and latitude coordinates of the geocoded result of an area will be (0,0). (Optional)
CENTER_LAT	NUMBER	Latitude value of the center of the area. (See the explanation for the CENTER_LONG column.) (Optional)
ROAD_SEGMENT_ID	NUMBER(10)	ID of the road segment to which the area center is set. This value must be set correctly if the geocoder is intended to work with the Oracle Spatial routing engine (described in Appendix C); otherwise, it can be set to any nonzero value, but it cannot be null. (Required)
COUNTRY_CODE_2	VARCHAR2(2)	2- letter ISO country code of the country to which the area belongs. (Required)
PARTITION_ID	NUMBER	Partition key used for partitioning geocoder data by geographic boundaries. If the data is not partitioned, set the value to 1. (Required)
NUM_STREETS	NUMBER	The number of streets inside this area. (Optional)

5.5.6 GC_ROAD_<suffix> Table

The GC_ROAD_<suffix> table (for example, GC_ROAD_US) stores road information for the country associated with the table name suffix. A road is a collection of road segments with the same name in the same settlement area; a road segment (defined in the GC_ROAD_SEGMENT_<suffix> table) is the segment of the road between two continuous intersections. The GC_ROAD_<suffix> table contains one or more rows for each road. (For example, it can contain multiple rows for a road if the road is associated with multiple settlements.) The GC_ROAD_<suffix> table contains the columns shown in [Table 5–11](#).

Table 5–11 GC_ROAD_<suffix> Table

Column Name	Data Type	Description
ROAD_ID	NUMBER	ID number of the road. (Required)
SETTLEMENT_ID	NUMBER(10)	ID number of the settlement to which the road belongs. (Required if the road is associated with a settlement)
MUNICIPALITY_ID	NUMBER(10)	ID number of the municipality to which the road belongs. (Required if the road is associated with a municipality)
PARENT_AREA_ID	NUMBER(10)	ID number of the parent area of the municipality to which the road belongs. (Required if the road is associated with a parent area)

Table 5–11 (Cont.) GC_ROAD_<suffix> Table

Column Name	Data Type	Description
LANG_CODE	VARCHAR2(3)	3-letter ISO national language code for the language for the road name. (Required)
NAME	VARCHAR2(64)	Name of the road, including the type (if any), the prefix (if any), and the suffix (if any). For example, N Main St as NAME, with Main as BASE_NAME. (Required)
BASE_NAME	VARCHAR2(64)	Name of the road, excluding the type (if any), the prefix (if any), and the suffix (if any). For example, N Main St as NAME, with Main as BASE_NAME. (Required)
PREFIX	VARCHAR2(32)	Prefix of the road name. For example, N Main St as NAME, with N as PREFIX. (Required if the road name has a prefix)
SUFFIX	VARCHAR2(32)	Suffix of the road name. For example, Main St NW as NAME, with NW as SUFFIX. (Required if the road name has a suffix)
STYPE_BEFORE	VARCHAR2(32)	Street type that precedes the base name. For example, Avenue Victor Hugo as NAME, with Avenue as STYPE_BEFORE and Victor Hugo as BASE_NAME. (Required if the road type precedes the base name)
STYPE_AFTER	VARCHAR2(32)	Street type that follows the base name. For example, Main St as NAME, with St as STYPE_AFTER and Main as BASE_NAME. (Required if the road type follows the base name)
STYPE_ATTACHED	VARCHAR2(1)	Contains T if the street type is in the same word with the street name; contains F if the street type is a separate word from the street name. For example, in a German street address of 123 Beethovenstrass, the street type is strass, and it is in the same word with the street name, which is Beethoven. (Required)
START_HN	NUMBER(5)	(Should be set to the same value as CENTER_HN; not currently used by Oracle)
CENTER_HN	NUMBER(5)	Leading numerical part of the center house number. The center house number is the left side house number at the start point of the center road segment, which is located in the center of the whole road. (See the explanation of house numbers after Table 5–12 in Section 5.5.7.) (Required)
END_HN	NUMBER(5)	(Should be set to the same value as CENTER_HN; not currently used by Oracle)
START_HN_SIDE	VARCHAR2(1)	(Should be set to the same value as CENTER_HN_SIDE; not currently used by Oracle)
CENTER_HN_SIDE	VARCHAR2(1)	Side of the road of the center house number: L for left or R for right. The center house number is the left side house number at the start point of the center road segment, which is located in the center of the whole road. (See the explanation of house numbers after Table 5–12 in Section 5.5.7.) (Required)
END_HN_SIDE	VARCHAR2(1)	(Should be set to the same value as CENTER_HN_SIDE; not currently used by Oracle)
START_LONG	NUMBER	(Should be set to the same value as CENTER_LONG; not currently used by Oracle)
START_LAT	NUMBER	(Should be set to the same value as CENTER_LAT; not currently used by Oracle)

Table 5–11 (Cont.) GC_ROAD_<suffix> Table

Column Name	Data Type	Description
CENTER_LONG	NUMBER	Longitude value of the center house number. The center house number is the left side house number at the start point of the center road segment, which is located in the center of the whole road. (See the explanation of house numbers after Table 5–12 in Section 5.5.7 .) (Required)
CENTER_LAT	NUMBER	Latitude value of the center house number. (See also the explanation of the CENTER_LONG column.) (Required)
END_LONG	NUMBER	(Should be set to the same value as CENTER_LONG; not currently used by Oracle)
END_LAT	NUMBER	(Should be set to the same value as CENTER_LAT; not currently used by Oracle)
START_ROAD_SEG_ID	NUMBER(5)	(Should be set to the same value as CENTER_ROAD_SEG_ID; not currently used by Oracle)
CENTER_ROAD_SEG_ID	NUMBER(5)	ID number of the road segment at the center point of the road. (Required)
END_ROAD_SEG_ID	NUMBER(5)	(Should be set to the same value as CENTER_ROAD_SEG_ID; not currently used by Oracle)
POSTAL_CODE	VARCHAR2(16)	Postal code for the road. (Required)
COUNTRY_CODE_2	VARCHAR2(2)	2- letter ISO country code of the country to which the road belongs. (Required)
PARTITION_ID	NUMBER	Partition key used for partitioning geocoder data by geographic boundaries. If the data is not partitioned, set the value to 1. (Required)
CENTER_HN2	VARCHAR2(10)	The second part of the center house number. (See the explanation of house numbers after Table 5–12 in Section 5.5.7 .) (Required)

5.5.7 GC_ROAD_SEGMENT_<suffix> Table

The GC_ROAD_SEGMENT_<suffix> table (for example, GC_ROAD_SEGMENT_US) stores road segment information for the country associated with the table name suffix. A road segment is the segment of the road between two continuous intersections, while a road (defined in the GC_ROAD_<suffix> table) is a collection of road segments with the same name in the same settlement area. The GC_ROAD_SEGMENT_<suffix> table contains one row for each road segment, and it contains the columns shown in [Table 5–12](#).

Table 5–12 GC_ROAD_SEGMENT_<suffix> Table

Column Name	Data Type	Description
ROAD_SEGMENT_ID	NUMBER	ID number of the road segment. (Required)
ROAD_ID	NUMBER	ID number of the road containing this road segment. (Required)
L_ADDR_FORMAT	VARCHAR2(1)	Left side address format. Specify N if there are one or more house numbers on the left side of the road segment; leave null if there is no house number on the left side of the road segment. (Required)

Table 5–12 (Cont.) GC_ROAD_SEGMENT_<suffix> Table

Column Name	Data Type	Description
R_ADDR_FORMAT	VARCHAR2(1)	Right side address format. Specify N if there are one or more house numbers on the right side of the road segment; leave null if there is no house number on the right side of the road segment. (Required)
L_ADDR_SCHEME	VARCHAR2(1)	Numbering scheme for house numbers on the left side of the road segment: O (all odd numbers), E (all even numbers), or M (mixture of odd and even numbers). (Required)
R_ADDR_SCHEME	VARCHAR2(1)	Numbering scheme for house numbers on the right side of the road segment: O (all odd numbers), E (all even numbers), or M (mixture of odd and even numbers). (Required)
START_HN	NUMBER(5)	The lowest house number on this road segment. (Required)
END_HN	NUMBER(5)	The highest house number on this road segment. (Required)
L_START_HN	NUMBER(5)	The leading numerical part of the left side starting house number. (See the explanation of house numbers after this table.) (Required)
L_END_HN	NUMBER(5)	The leading numerical part of the left side ending house number. (See the explanation of house numbers after this table.) (Required)
R_START_HN	NUMBER(5)	The leading numerical part of the right side starting house number. (See the explanation of house numbers after this table.) (Required)
R_END_HN	NUMBER(5)	The leading numerical part of the right side ending house number. (See the explanation of house numbers after this table.) (Required)
POSTAL_CODE	VARCHAR2(16)	Postal code for the road segment. If the left side and right side of the road segment belong to two different postal codes, create two rows for the road segment with identical values in all columns except for POSTAL_CODE. (Required)
GEOMETRY	SDO_GEOMETRY	Spatial geometry object representing the road segment. (Required)
COUNTRY_CODE_2	VARCHAR2(2)	2- letter ISO country code of the country to which the road segment belongs. (Required)
PARTITION_ID	NUMBER	Partition key used for partitioning geocoder data by geographic boundaries. If the data is not partitioned, set the value to 1. (Required)
L_START_HN2	VARCHAR2(10)	The second part of the left side starting house number. (See the explanation of house numbers after this table.) (Required if the left side starting house number has a second part)
L_END_HN2	VARCHAR2(10)	The second part of the left side ending house number. (See the explanation of house numbers after this table.) (Required if the left side ending house number has a second part)
R_START_HN2	VARCHAR2(10)	The second part of the right side starting house number. (See the explanation of house numbers after this table.) (Required if the right side starting house number has a second part)

Table 5–12 (Cont.) GC_ROAD_SEGMENT_<suffix> Table

Column Name	Data Type	Description
R_END_HN2	VARCHAR2(10)	The second part of the right side ending house number. (See the explanation of house numbers after this table.) (Required if the right side ending house number has a second part)

The starting house number is the house number at the starting point of the street segment, which is the first shape point of the road segment geometry (GEOMETRY column). The ending house number is the house number at the ending point of the street segment, which is the last shape point of the road segment geometry. The left and right side starting house numbers do not have to be lower than the left and right ending house numbers.

A house number is divided into two parts: the leading numerical part and the second part, which is the rest of the house number. The leading numerical part is the numerical part of the house number that starts from the beginning of the whole house number string and ends just before the first non-numeric character (if any). If the house number contains any non-numeric characters, the second part of the house number is the part from the first non-numeric character through the last character. For example, if the house number is 123, the leading numerical part is 123 and the second part is null; however, if the house number is 123A23, the leading numerical part is 123 and the second part is A23.

Coordinate Systems (Spatial Reference Systems)

This chapter describes in greater detail the Oracle Spatial coordinate system support, which was introduced in [Section 1.5.4](#). You can store and manipulate SDO_GEOMETRY objects in a variety of coordinate systems.

For reference information about coordinate system transformation functions and procedures in the MDSYS.SDO_CS package, see [Chapter 13](#).

This chapter contains the following major sections:

- [Section 6.1, "Terms and Concepts"](#)
- [Section 6.2, "Geodetic Coordinate Support"](#)
- [Section 6.3, "Local Coordinate Support"](#)
- [Section 6.4, "EPSG Model and Spatial"](#)
- [Section 6.5, "TFM_PLAN Object Type"](#)
- [Section 6.6, "Coordinate Systems Data Structures"](#)
- [Section 6.7, "Legacy Tables and Views"](#)
- [Section 6.8, "Creating a User-Defined Coordinate Reference System"](#)
- [Section 6.9, "Notes and Restrictions with Coordinate Systems Support"](#)
- [Section 6.10, "U.S. National Grid Support"](#)
- [Section 6.11, "Example of Coordinate System Transformation"](#)

6.1 Terms and Concepts

This section explains important terms and concepts related to coordinate system support in Oracle Spatial.

6.1.1 Coordinate System (Spatial Reference System)

A **coordinate system** (also called a *spatial reference system*) is a means of assigning coordinates to a location and establishing relationships between sets of such coordinates. It enables the interpretation of a set of coordinates as a representation of a position in a real world space.

The term **coordinate reference system** has the same meaning as coordinate system for Spatial, and the terms are used interchangeably. European Petroleum Survey Group (EPSG) specifications and documentation typically use the term coordinate reference

system. (EPSG has its own meaning for the term *coordinate system*, as noted in [Section 6.6.11](#).)

6.1.2 Cartesian Coordinates

Cartesian coordinates are coordinates that measure the position of a point from a defined origin along axes that are perpendicular in the represented two-dimensional or three-dimensional space.

6.1.3 Geodetic Coordinates (Geographic Coordinates)

Geodetic coordinates (sometimes called *geographic coordinates*) are angular coordinates (longitude and latitude), closely related to spherical polar coordinates, and are defined relative to a particular Earth geodetic datum (described in [Section 6.1.6](#)). For more information about geodetic coordinate support, see [Section 6.2](#).

6.1.4 Projected Coordinates

Projected coordinates are planar Cartesian coordinates that result from performing a mathematical mapping from a point on the Earth's surface to a plane. There are many such mathematical mappings, each used for a particular purpose.

6.1.5 Local Coordinates

Local coordinates are Cartesian coordinates in a non-Earth (non-georeferenced) coordinate system. [Section 6.3](#) describes local coordinate support in Spatial.

6.1.6 Geodetic Datum

A **geodetic datum** (or **datum**) is a means of shifting and rotating an ellipsoid to represent the figure of the Earth, usually as an oblate spheroid, that approximates the surface of the Earth locally or globally, and is the reference for the system of geodetic coordinates.

Each geodetic coordinate system is based on a datum.

6.1.7 Transformation

Transformation is the conversion of coordinates from one coordinate system to another coordinate system.

If the coordinate system is georeferenced, transformation can involve datum transformation: the conversion of geodetic coordinates from one geodetic datum to another geodetic datum, usually involving changes in the shape, orientation, and center position of the reference ellipsoid.

6.2 Geodetic Coordinate Support

Effective with Oracle9i, Spatial provides a rational and complete treatment of geodetic coordinates. Before Oracle9i, Spatial computations were based solely on flat (Cartesian) coordinates, regardless of the coordinate system specified for the layer of geometries. Consequently, computations for data in geodetic coordinate systems were inaccurate, because they always treated the coordinates as if they were on a flat surface, and they did not consider the curvature of the surface.

Effective with release 9.2, ellipsoidal surface computations consider the curvatures of the Earth in the specified geodetic coordinate system and return correct, accurate results. In other words, Spatial queries return the right answers all the time.

6.2.1 Geodesy and Two-Dimensional Geometry

A two-dimensional geometry is a surface geometry, but the important question is: What is the *surface*? A flat surface (plane) is accurately represented by Cartesian coordinates. However, Cartesian coordinates are not adequate for representing the surface of a solid. A commonly used surface for spatial geometry is the surface of the Earth, and the laws of geometry there are different than they are in a plane. For example, on the Earth's surface there are no parallel lines: lines are geodesics, and all geodesics intersect. Thus, closed curved surface problems cannot be done accurately with Cartesian geometry.

Spatial provides accurate results regardless of the coordinate system or the size of the area involved, without requiring that the data be projected to a flat surface. The results are accurate regardless of where on the Earth's surface the query is focused, even in "special" areas such as the poles. Thus, you can store coordinates in any datum and projections that you choose, and you can perform accurate queries regardless of the coordinate system.

6.2.2 Choosing a Geodetic or Projected Coordinate System

For applications that deal with the Earth's surface, the data can be represented using a geodetic coordinate system or a projected plane coordinate system. In deciding which approach to take with the data, consider any needs related to accuracy and performance:

- Accuracy

For many spatial applications, the area is sufficiently small to allow adequate computations on Cartesian coordinates in a local projection. For example, the New Hampshire State Plane local projection provides adequate accuracy for most spatial applications that use data for that state.

However, Cartesian computations on a plane projection will never give accurate results for a large area such as Canada or Scandinavia. For example, a query asking if Stockholm, Sweden and Helsinki, Finland are within a specified distance may return an incorrect result if the specified distance is close to the actual measured distance. Computations involving large areas or requiring very precise accuracy must account for the curvature of the Earth's surface.

- Performance

Spherical computations use more computing resources than Cartesian computations. Some operations using geodetic coordinates may take longer to complete than the same operations using Cartesian coordinates.

6.2.3 Geodetic MBRs

To create a query window for certain operations on geodetic data, use an MBR (minimum bounding rectangle) by specifying an SDO_ETYPE value of 1003 or 2003 and an SDO_INTERPRETATION value of 3, as described in [Table 2-2](#) in [Section 2.2.4](#). A geodetic MBR can be used with the following operators: [SDO_FILTER](#), [SDO_RELATE](#) with the ANYINTERACT mask, [SDO_ANYINTERACT](#), and [SDO_WITHIN_DISTANCE](#).

[Example 6-1](#) requests the names of all cola markets that are likely to interact spatially with a geodetic MBR.

Example 6-1 Using a Geodetic MBR

```
SELECT c.name FROM cola_markets_cs c WHERE
  SDO_FILTER(c.shape,
    SDO_GEOMETRY(
      2003,
      8307,      -- SRID for WGS 84 longitude/latitude
      NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(6,5, 10,10)
    ) = 'TRUE';
```

[Example 6-1](#) produces the following output (assuming the data as defined in [Example 6-7](#) in [Section 6.11](#)):

```
NAME
-----
cola_c
cola_b
cola_d
```

The following considerations apply to the use of geodetic MBRs:

- Do not use a geodetic MBR with spatial objects stored in the database. Use it only to construct a query window.
- The lower-left Y coordinate (minY) must be less than the upper-right Y coordinate (maxY). If the lower-left X coordinate (minX) is greater than the upper-right X coordinate (maxX), the window is assumed to cross the date line meridian (that is, the meridian "opposite" the prime meridian, or both 180 and -180 longitude). For example, an MBR of (-10,10, -100, 20) with longitude/latitude data goes three-fourths of the way around the Earth (crossing the date line meridian), and goes from latitude lines 10 to 20.
- When Spatial constructs the MBR internally for the query, lines along latitude lines are densified by adding points at one-degree intervals. This might affect results for objects within a few meters of the edge of the MBR (especially objects in the middle latitudes in both hemispheres).

The following additional examples show special or unusual cases, to illustrate how a geodetic MBR is interpreted with longitude/latitude data:

- (10,0, -110,20) crosses the date line meridian and goes most of the way around the world, and goes from the equator to latitude 20.
- (10,-90, 40,90) is a band from the South Pole to the North Pole between longitudes 10 and 40.
- (10,-90, 40,50) is a band from the South Pole to latitude 50 between longitudes 10 and 40.
- (-180,-10, 180,5) is a band that wraps the equator from 10 degrees south to 5 degrees north.
- (-180,-90, 180,90) is the whole Earth.
- (-180,-90, 180,50) is the whole Earth below latitude 50.
- (-180,50, 180,90) is the whole Earth above latitude 50.

6.2.4 Other Considerations and Requirements with Geodetic Data

The following geometries are not permitted if a geodetic coordinate system is used:

- Circles
- Circular arcs

Geodetic coordinate system support is provided only for geometries that consist of points or geodesics (lines on the ellipsoid). If you have geometries containing circles or circular arcs in a projected coordinate system, you can densify them using the `SDO_GEOM.SDO_ARC_DENSIFY` function (documented in [Chapter 15](#)) before transforming them to geodetic coordinates, and then perform Spatial operations on the resulting geometries.

The following size limits apply with geodetic data:

- No polygon element can have an area larger than one-half the surface of the Earth.
- In a line, the distance between two adjacent coordinates cannot be greater than or equal to one-half the perimeter (a great circle) of the Earth.

If you need to work with larger elements, first break these elements into multiple smaller elements and work with them. For example, you cannot create a geometry representing the entire ocean surface of the Earth; however, you can create multiple geometries, each representing part of the overall ocean surface. To work with a line string that is greater than or equal to one-half the perimeter of the Earth, you can add one or more intermediate points on the line so that all adjacent coordinates are less than one-half the perimeter of the Earth.

To take full advantage of Spatial features, you must index geodetic data layers using a geodetic R-tree index. (You can create a non-geodetic R-tree or quadtree index on geodetic data by specifying `'geodetic=FALSE'` in the PARAMETERS clause of the `CREATE INDEX` statement; however, this is not recommended. See the Usage Notes for the `CREATE INDEX` statement in [Chapter 10](#) for more information.) In addition, for Spatial release 9.0.1 and later you must delete (`DROP INDEX`) and re-create all spatial indexes on geodetic data from a release before 9.0.1.

Tolerance is specified as meters for geodetic layers. If you use tolerance values that are typical for non-geodetic data, these values are interpreted as meters for geodetic data. For example, if you specify a tolerance value of 0.05 for geodetic data, this is interpreted as precise to 5 centimeters. If this value is more precise than your applications need, performance may be affected because of the internal computational steps taken to implement the specified precision. (For more information about tolerance, see [Section 1.5.5](#).)

For geodetic layers, you must specify the dimensional extents in the index metadata as -180,180 for longitude and -90,90 for latitude. The following statement (from [Example 6-7](#) in [Section 6.11](#)) specifies these extents (with a 10-meter tolerance value in each dimension) for a geodetic data layer:

```
INSERT INTO user_sdo_geom_metadata
  (TABLE_NAME,
   COLUMN_NAME,
   DIMINFO,
   SRID)
VALUES (
  'cola_markets_cs',
  'shape',
  SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT('Longitude', -180, 180, 10), -- 10 meters tolerance
    SDO_DIM_ELEMENT('Latitude', -90, 90, 10) -- 10 meters tolerance
```

```
),  
 8307 -- SRID for 'Longitude / Latitude (WGS 84)' coordinate system  
);
```

See [Section 6.9](#) for additional notes and restrictions relating to geodetic data.

6.3 Local Coordinate Support

Spatial provides a level of support for local coordinate systems. Local coordinate systems are often used in CAD systems, and they can also be used in local surveys where the relationship between the surveyed site and the rest of the world is not important.

Several local coordinate systems are predefined and included with Spatial in the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)). These supplied local coordinate systems, whose names start with *Non-Earth*, define non-Earth Cartesian coordinate systems based on different units of measurement (*Meter*, *Millimeter*, *Inch*, and so on).

In the current release, you cannot perform coordinate system transformation between local and Earth-based coordinate systems; and when transforming a geometry or layer of geometries between local coordinate systems, you can only to convert coordinates in a local coordinate system from one unit of measurement to another (for example, inches to millimeters). However, you can perform all other Spatial operations (for example, using [SDO_RELATE](#), [SDO_WITHIN_DISTANCE](#), and other operators) with local coordinate systems.

6.4 EPSG Model and Spatial

Effective with Oracle Database 10g release 2 (10.2), the Oracle Spatial coordinate system support is based on, but is not always identical to, the European Petroleum Survey Group (EPSG) data model and data set (described in detail at <http://www.epsg.org>). This approach provides the benefits of standardization, expanded support, and flexibility:

- The EPSG model is a comprehensive and widely accepted standard for data representation, so users familiar with it can more easily understand Spatial storage and operations.
- Support is provided for more coordinate systems and their associated datums, ellipsoids, and projections. For example, some of the EPSG geographic and projected coordinate systems had no counterpart among coordinate systems supported for previous Spatial releases. Their addition results in an expanded set of supported coordinate systems.
- Data transformations are more flexible. Instead of there being only one possible Oracle-defined transformation path between a given source and target coordinate system, you can specify alternative paths to be used for a specific area of applicability (that is, use case) or as the systemwide default.

The rest of this section describes this flexibility.

For data transformations (that is, transforming data from one coordinate system to another), you can now control which transformation rules are to be applied. In previous releases, and in the current release by default, Spatial performs transformations based only on the specified source and target coordinate systems, using predetermined intermediate transformation steps. The assumption behind that default approach is that there is a single correct or preferable transformation chain.

By default, then, Spatial applies certain transformation methods for each supported transformation between specific pairs of source and target coordinate systems. For example, there are over 500 supported transformations from specific coordinate systems to the WGS 84 (longitude/latitude) coordinate system, which has the EPSG SRID value of 4326. As one example, for a transformation from SRID 4605 to SRID 4326, Spatial can use the transformation method with the COORD_OP_ID value 1445, as indicated in the SDO_COORD_OPS table (described in [Section 6.6.8](#)), which contains one row for each transformation operation between coordinate systems.

However, you can override the default transformation by specifying a different method (from the set of Oracle-supplied methods) for the transformation for any given source and target SRID combination. You can specify a transformation as the new systemwide default, or you can associate the transformation with a named use case that can be specified when transforming a layer of spatial geometries. (A **use case** is simply a name given to a usage scenario or area of applicability, such as *Project XYZ* or *Mike's Favorite Transformations*; there is no relationship between use cases and database users or schemas.)

To specify a transformation as either the systemwide default or associated with a use case, use the [SDO_CS.ADD_PREFERENCE_FOR_OP](#) procedure. To remove a previously specified preference, use the [SDO_CS.REVOKE_PREFERENCE_FOR_OP](#) procedure.

When it performs a coordinate system transformation, Spatial follows these general steps to determine the specific transformation to use:

1. If a use case has been specified, the transformation associated with that use case is applied.
2. If no use case has been specified and if a user-defined systemwide transformation has been created for the specified source and target coordinate system pair, that transformation is applied.
3. If no use case has been specified and if no user-defined transformation exists for the specified source and target coordinate system pair, the behavior depends on whether or not EPSG rules have been created, such as by the [SDO_CS.CREATE_OBVIOUS_EPSG_RULES](#) procedure:
 - If the EPSG rules have been created and if an EPSG rule is defined for this transformation, the EPSG transformation is applied.
 - If the EPSG rules have not been created, or if they have been created but no EPSG rule is defined for this transformation, the Oracle Spatial default transformation is applied.

6.5 TFM_PLAN Object Type

The object type TFM_PLAN is used by several SDO_CS package subprograms to specify a transformation plan. For example, to create a concatenated operation that consists of two operations specified by a parameter of type TFM_PLAN, use the [SDO_CS.CREATE_CONCATENATED_OP](#) procedure.

Oracle Spatial defines the object type TFM_PLAN as:

```
CREATE TYPE tfm_plan AS OBJECT (
  THE_PLAN SDO_TFM_CHAIN);
```

The SDO_TFM_CHAIN type is defined as VARRAY(1048576) OF NUMBER.

Within the SDO_TFM_CHAIN array:

- The first element specifies the SRID of the source coordinate system.
- Each pair of elements after the first element specifies an operation ID and the SRID of a target coordinate system.

6.6 Coordinate Systems Data Structures

The coordinate systems functions and procedures use information provided in the tables and views supplied with Oracle Spatial. The tables and views are part of the MDSYS schema; however, public synonyms are defined, so you do not need to specify *MDSYS.* before the table or view name. The definitions and data in these tables and views are based on the EPSG data model and data set, as explained in [Section 6.4](#).

The coordinate system tables fit into several general categories:

- Coordinate system general information: SDO_COORD_SYS, SDO_COORD_REF_SYS
- Elements or aspects of a coordinate system definition: SDO_DATUMS, SDO_ELLIPSOIDS, SDO_PRIME_MERIDIANS
- Datum transformation support: SDO_COORD_OPS, SDO_COORD_OP_METHODS, SDO_COORD_OP_PARAM_USE, SDO_COORD_OP_PARAM_VALS, SDO_COORD_OP_PARAMS, SDO_COORD_OP_PATHS, SDO_PREFERRED_OPS_SYSTEM, SDO_PREFERRED_OPS_USER
- Others related to coordinate system definition: SDO_COORD_AXES, SDO_COORD_AXIS_NAMES, SDO_UNITS_OF_MEASURE

Several views are provided that are identical to or subsets of coordinate system tables:

- SDO_COORD_REF_SYSTEM, which contains the same columns as the SDO_COORD_REF_SYS table. Use the SDO_COORD_REF_SYSTEM view instead of the COORD_REF_SYS table for any insert, update, or delete operations.
- Subsets of SDO_DATUMS, selected according to the value in the DATUM_TYPE column: SDO_DATUM_ENGINEERING, SDO_DATUM_GEODETTIC, SDO_DATUM_VERTICAL.
- Subsets of SDO_COORD_REF_SYS, selected according to the value in the COORD_REF_SYS_KIND column: SDO_CRS_COMPOUND, SDO_CRS_ENGINEERING, SDO_CRS_GEOCENTRIC, SDO_CRS_GEOGRAPHIC2D, SDO_CRS_GEOGRAPHIC3D, SDO_CRS_PROJECTED, SDO_CRS_VERTICAL.

The rest of this section explains these tables and views, in alphabetical order. (Many column descriptions are adapted or taken from EPSG descriptions.)

In addition to the tables and views in this section, Spatial provides several legacy tables whose definitions and data match those of certain Spatial system tables used in previous releases. [Section 6.7](#) describes the legacy tables.

Note: You should not modify or delete any Oracle-supplied information in any of the tables or views that are used for coordinate system support.

If you want to create a user-defined coordinate system, see [Section 6.8](#).

6.6.1 SDO_COORD_AXES Table

The SDO_COORD_AXES table contains one row for each coordinate system axis definition. This table contains the columns shown in [Table 6-1](#).

Table 6–1 SDO_COORD_AXES Table

Column Name	Data Type	Description
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system to which this axis applies.
COORD_AXIS_NAME_ID	NUMBER(10)	ID number of a coordinate system axis name. Matches a value in the COORD_AXIS_NAME_ID column of the SDO_COORD_AXIS_NAMES table (described in Section 6.6.2). Example: 9901 (for Geodetic latitude)
COORD_AXIS_ORIENTATION	VARCHAR2(24)	The direction of orientation for the coordinate system axis. Example: east
COORD_AXIS_ABBREVIATION	VARCHAR2(24)	The abbreviation for the coordinate system axis orientation. Example: E
UOM_ID	NUMBER(10)	ID number of the unit of measurement associated with the axis. Matches a value in the UOM_ID column of the SDO_UNITS_OF_MEASURE table (described in Section 6.6.27).
ORDER	NUMBER(10)	Position of this axis within the coordinate system (1, 2, or 3).

6.6.2 SDO_COORD_AXIS_NAMES Table

The SDO_COORD_AXIS_NAMES table contains one row for each axis that can be used in a coordinate system definition. This table contains the columns shown in [Table 6–2](#).

Table 6–2 SDO_COORD_AXIS_NAMES Table

Column Name	Data Type	Description
COORD_AXIS_NAME_ID	NUMBER(10)	ID number of the coordinate axis name. Example: 9926
COORD_AXIS_NAME	VARCHAR2(80)	Name of the coordinate axis. Example: Spherical latitude

6.6.3 SDO_COORD_OP_METHODS Table

The SDO_COORD_OP_METHODS table contains one row for each coordinate systems transformation method. This table contains the columns shown in [Table 6–3](#).

Table 6–3 SDO_COORD_OP_METHODS Table

Column Name	Data Type	Description
COORD_OP_METHOD_ID	NUMBER(10)	ID number of the coordinate system transformation method. Example: 9613
COORD_OP_METHOD_NAME	VARCHAR2(50)	Name of the method. Example: NADCON
REVERSE_OP	NUMBER(1)	Contains 1 if reversal of the transformation (from the current target coordinate system to the source coordinate system) can be achieved by reversing the sign of each parameter value; contains 0 if a separate operation must be defined for reversal of the transformation.

Table 6–3 (Cont.) SDO_COORD_OP_METHODS Table

Column Name	Data Type	Description
INFORMATION_SOURCE	VARCHAR2(254)	Origin of this information. Example: US Coast and geodetic Survey - http://www.ngs.noaa.gov
DATA_SOURCE	VARCHAR2(40)	Organization providing the data for this record. Example: EPSG

6.6.4 SDO_COORD_OP_PARAM_USE Table

The SDO_COORD_OP_PARAM_USE table contains one row for each combination of transformation method and transformation operation parameter that is available for use. This table contains the columns shown in [Table 6–4](#).

Table 6–4 SDO_COORD_OP_PARAM_USE Table

Column Name	Data Type	Description
COORD_OP_METHOD_ID	NUMBER(10)	ID number of the coordinate system transformation method. Matches a value in the COORD_OP_METHOD_ID column of the COORD_OP_METHODS table (described in Section 6.6.3).
PARAMETER_ID	NUMBER(10)	ID number of the parameter for transformation operations. Matches a value in the PARAMETER_ID column of the SDO_COORD_OP_PARAMS table (described in Section 6.6.6).
SORT_ORDER	NUMBER(5)	A number indicating the position of this parameter in the sequence of parameters for this method. Example: 2 for the second parameter
PARAM_SIGN_REVERSAL	VARCHAR2(3)	Yes if reversal of the transformation (from the current target coordinate system to the source coordinate system) can be achieved by reversing the sign of each parameter value; No if a separate operation must be defined for reversal of the transformation.

6.6.5 SDO_COORD_OP_PARAM_VALS Table

The SDO_COORD_OP_PARAM_VALS table contains information about parameter values for each coordinate system transformation method. This table contains the columns shown in [Table 6–5](#).

Table 6–5 SDO_COORD_OP_PARAM_VALS Table

Column Name	Data Type	Description
COORD_OP_ID	NUMBER(10)	ID number of the coordinate transformation operation. Matches a value in the COORD_OP_ID column of the SDO_COORD_OPS table (described in Section 6.6.8).
COORD_OP_METHOD_ID	NUMBER(10)	Coordinate operation method ID. Must match a COORD_OP_METHOD_ID value in the SDO_COORD_OP_METHODS table (see Section 6.6.3).
PARAMETER_ID	NUMBER(10)	ID number of the parameter for transformation operations. Matches a value in the PARAMETER_ID column of the SDO_COORD_OP_PARAMS table (described in Section 6.6.6).

Table 6–5 (Cont.) SDO_COORD_OP_PARAM_VALS Table

Column Name	Data Type	Description
PARAMETER_VALUE	FLOAT(49)	Value of the parameter for this operation.
PARAM_VALUE_FILE_REF	VARCHAR2(254)	Name of the file containing the value data, if a single value for the parameter is not sufficient.
UOM_ID	NUMBER(10)	ID number of the unit of measurement associated with the operation. Matches a value in the UOM_ID column of the SDO_UNITS_OF_MEASURE table (described in Section 6.6.27).

6.6.6 SDO_COORD_OP_PARAMS Table

The SDO_COORD_OP_PARAMS table contains one row for each available parameter for transformation operations. This table contains the columns shown in [Table 6–6](#).

Table 6–6 SDO_COORD_OP_PARAMS Table

Column Name	Data Type	Description
PARAMETER_ID	NUMBER(10)	ID number of the parameter. Example: 8608
PARAMETER_NAME	VARCHAR2(80)	Name of the operation. Example: X-axis rotation
INFORMATION_SOURCE	VARCHAR2(254)	Origin of this information. Example: EPSG guidance note number 7.
DATA_SOURCE	VARCHAR2(40)	Organization providing the data for this record. Example: EPSG

6.6.7 SDO_COORD_OP_PATHS Table

The SDO_COORD_OP_PATHS table contains one row for each atomic step in a concatenated operation. This table contains the columns shown in [Table 6–7](#).

Table 6–7 SDO_COORD_OP_PATHS Table

Column Name	Data Type	Description
CONCAT_OPERATION_ID	NUMBER(10)	ID number of the concatenation operation. Must match a COORD_OP_ID value in the SDO_COORD_OPS table (described in Section 6.6.8) for which the COORD_OP_TYPE value is CONCATENATION.
SINGLE_OPERATION_ID	NUMBER(10)	ID number of the single coordinate operation for this step (atomic operation) in a concatenated operation. Must match a COORD_OP_ID value in the SDO_COORD_OPS table (described in Section 6.6.8).
SINGLE_OP_SOURCE_ID	NUMBER(10)	ID number of source coordinate reference system for the single coordinate operation for this step. Must match an SRID value in the SDO_COORD_REF_SYS table (described in Section 6.6.9).
SINGLE_OP_TARGET_ID	NUMBER(10)	ID number of target coordinate reference system for the single coordinate operation for this step. Must match an SRID value in the SDO_COORD_REF_SYS table (described in Section 6.6.9).
OP_PATH_STEP	NUMBER(5)	Sequence number of this step (atomic operation) within this concatenated operation.

6.6.8 SDO_COORD_OPS Table

The SDO_COORD_OPS table contains one row for each transformation operation between coordinate systems. This table contains the columns shown in [Table 6–8](#).

Table 6–8 SDO_COORD_OPS Table

Column Name	Data Type	Description
COORD_OP_ID	NUMBER(10)	ID number of the coordinate transformation operation. Example: 101
COORD_OP_NAME	VARCHAR2(80)	Name of the operation. Example: ED50 to WGS 84 (14)
COORD_OP_TYPE	VARCHAR2(24)	Type of operation. One of the following: CONCATENATED OPERATION, CONVERSION, or TRANSFORMATION
SOURCE_SRID	NUMBER(10)	SRID of the coordinate system from which to perform the transformation. Example: 4230
TARGET_SRID	NUMBER(10)	SRID of the coordinate system into which to perform the transformation. Example: 4326
COORD_TFM_VERSION	VARCHAR2(24)	Name assigned by EPSG to the coordinate transformation. Example: 5Nat -NSea90
COORD_OP_VARIANT	NUMBER(5)	A variant of the more generic method specified in COORD_OP_METHOD_ID. Example: 14
COORD_OP_METHOD_ID	NUMBER(10)	Coordinate operation method ID. Must match a COORD_OP_METHOD_ID value in the SDO_COORD_OP_METHODS table (see Section 6.6.3). Several operations can use a method. Example: 9617
UOM_ID_SOURCE_OFFSETS	NUMBER(10)	ID number of the unit of measurement for offsets in the source coordinate system. Matches a value in the UOM_ID column of the SDO_UNITS_OF_MEASURE table (described in Section 6.6.27).
UOM_ID_TARGET_OFFSETS	NUMBER(10)	ID number of the unit of measurement for offsets in the target coordinate system. Matches a value in the UOM_ID column of the SDO_UNITS_OF_MEASURE table (described in Section 6.6.27).
INFORMATION_SOURCE	VARCHAR2(254)	Origin of this information. Example: Institut de Geomatica; Barcelona
DATA_SOURCE	VARCHAR2(40)	Organization providing the data for this record. Example: EPSG
SHOW_OPERATION	NUMBER(3)	(Currently not used.)
IS_LEGACY	VARCHAR2(5)	TRUE if the operation was included in Oracle Spatial before release 10.2; FALSE if the operation is new in Oracle Spatial release 10.2.
LEGACY_CODE	NUMBER(10)	For any EPSG coordinate transformation operation that has a semantically identical legacy (in Oracle Spatial before release 10.2) counterpart, the COORD_OP_ID value of the legacy coordinate transformation operation.

Table 6–8 (Cont.) SDO_COORD_OPS Table

Column Name	Data Type	Description
REVERSE_OP	NUMBER(1)	Contains 1 if reversal of the transformation (from the current target coordinate system to the source coordinate system) is defined as achievable by reversing the sign of each parameter value; contains 0 if a separate operation must be defined for reversal of the transformation. If REVERSE_OP contains 1, the operations that are actually implemented are indicated by the values for IS_IMPLEMENTED_FORWARD and IS_IMPLEMENTED_REVERSE.
IS_IMPLEMENTED_FORWARD	NUMBER(1)	Contains 1 if the forward operation is implemented; contains 0 if the forward operation is not implemented.
IS_IMPLEMENTED_REVERSE	NUMBER(1)	Contains 1 if the reverse operation is implemented; contains 0 if the reverse operation is not implemented.

6.6.9 SDO_COORD_REF_SYS Table

The SDO_COORD_REF_SYS table contains one row for each coordinate reference system. This table contains the columns shown in [Table 6–9](#). (The SDO_COORD_REF_SYS table is roughly patterned after the EPSG Coordinate Reference System table.)

Note: If you need to perform an insert, update, or delete operation, you *must* perform it on the SDO_COORD_REF_SYSTEM view, which contains the same columns as the SDO_COORD_REF_SYS table. The SDO_COORD_REF_SYSTEM view is described in [Section 6.6.10](#).

Table 6–9 SDO_COORD_REF_SYS Table

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system. Example: 8307
COORD_REF_SYS_NAME	VARCHAR2(80)	Name of the coordinate reference system. Example: Longitude / Latitude (WGS 84)
COORD_REF_SYS_KIND	VARCHAR2(24)	Category for the coordinate system. Example: GEOGRAPHIC2D
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system used for the coordinate reference system. Must match a COORD_SYS_ID value in the SDO_COORD_SYS table (see Section 6.6.11).
DATUM_ID	NUMBER(10)	ID number of the datum used for the coordinate reference system. Null for a projected coordinate system. For a geodetic coordinate system, must match a DATUM_ID value in the SDO_DATUMS table (see Section 6.6.22). Example: 10115
GEOG_CRS_DATUM_ID	NUMBER(10)	ID number of the datum used for the coordinate reference system. For a projected coordinate system, must match the DATUM_ID value (in the SDO_DATUMS table, described in Section 6.6.22) of the geodetic coordinate system on which the projected coordinate system is based. For a geodetic coordinate system, must match the DATUM_ID value. Example: 10115

Table 6–9 (Cont.) SDO_COORD_REF_SYS Table

Column Name	Data Type	Description
SOURCE_GEOG_SRID	NUMBER(10)	For a projected coordinate reference system, the ID number for the associated geodetic coordinate system.
PROJECTION_CONV_ID	NUMBER(10)	For a projected coordinate reference system, the COORD_OP_ID value of the conversion operation used to convert the projected coordinated system to and from the source geographic coordinate system.
CMPD_HORIZ_SRID	NUMBER(10)	(EPSG-assigned value; not used by Oracle Spatial. The EPSG description is: "For compound CRS only, the code of the horizontal component of the Compound CRS.")
CMPD_VERT_SRID	NUMBER(10)	(EPSG-assigned value; not used by Oracle Spatial. The EPSG description is: "For compound CRS only, the code of the vertical component of the Compound CRS.")
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).
IS_LEGACY	VARCHAR2(5)	TRUE if the coordinate system definition was included in Oracle Spatial before release 10.2; FALSE if the coordinate system definition is new in Oracle Spatial release 10.2.
LEGACY_CODE	NUMBER(10)	For any EPSG coordinate reference system that has a semantically identical legacy (in Oracle Spatial before release 10.2) counterpart, the SRID value of the legacy coordinate system.
LEGACY_WKTEXT	VARCHAR2(2046)	If IS_LEGACY is TRUE, contains the well-known text description of the coordinate system. Example: GEOGCS ["Longitude / Latitude (WGS 84)", DATUM ["WGS 84", SPHEROID ["WGS 84", 6378137, 298.257223563]], PRIMEM ["Greenwich", 0.000000], UNIT ["Decimal Degree", 0.01745329251994330]]
LEGACY_CS_BOUNDS	SDO_GEOMETRY	For a legacy coordinate system, the dimensional boundary (if any).
IS_VALID	VARCHAR2(5)	TRUE if the EPSG record for the coordinate reference system is completely defined; FALSE if the EPSG record for the coordinate reference system is not completely defined.
SUPPORTS_SDO_GEOMETRY	VARCHAR2(5)	TRUE if the COORD_REF_SYS_KIND column contains ENGINEERING, GEOGRAPHIC2D, or PROJECTED CRS; FALSE if the COORD_REF_SYS_KIND column contains any other value.

See also the information about the following views that are defined based on the value of the COORD_REF_SYS_KIND column:

- SDO_CRS_COMPOUND (Section 6.6.12)
- SDO_CRS_ENGINEERING (Section 6.6.13)
- SDO_CRS_GEOCENTRIC (Section 6.6.14)

- SDO_CRS_GEOGRAPHIC2D (Section 6.6.15)
- SDO_CRS_GEOGRAPHIC3D (Section 6.6.16)
- SDO_CRS_PROJECTED (Section 6.6.17)
- SDO_CRS_VERTICAL (Section 6.6.18)

6.6.10 SDO_COORD_REF_SYSTEM View

The SDO_COORD_REF_SYSTEM view contains the same columns as the SDO_COORD_REF_SYS table, which is described in Section 6.6.9. However, the SDO_COORD_REF_SYSTEM view has a trigger defined on it, so that any insert, update, or delete operations performed on the view cause all relevant Spatial system tables to have the appropriate operations performed on them.

Therefore, if you need to perform an insert, update, or delete operation, you *must* perform it on the SDO_COORD_REF_SYSTEM view, *not* the SDO_COORD_REF_SYS table.

6.6.11 SDO_COORD_SYS Table

The SDO_COORD_SYS table contains rows with information about coordinate systems. This table contains the columns shown in Table 6–10. (The SDO_COORD_SYS table is roughly patterned after the EPSG Coordinate System table, where a coordinate system is described as "a pair of reusable axes.")

Table 6–10 SDO_COORD_SYS Table

Column Name	Data Type	Description
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system. Example: 6405
COORD_SYS_NAME	VARCHAR2(254)	Name of the coordinate system. Example: Ellipsoidal 2D CS. Axes: latitude, longitude. Orientations: north, east. UoM: dec deg
COORD_SYS_TYPE	VARCHAR2(24)	Type of coordinate system. Example: ellipsoidal
DIMENSION	NUMBER(5)	Number of dimensions represented by the coordinate system.
INFORMATION_SOURCE	VARCHAR2(254)	Origin of this information.
DATA_SOURCE	VARCHAR2(40)	Organization providing the data for this record.

6.6.12 SDO_CRS_COMPOUND View

The SDO_CRS_COMPOUND view contains selected information from the SDO_COORD_REF_SYS table (described in Section 6.6.9) where the COORD_REF_SYS_KIND column value is COMPOUND. This view contains the columns shown in Table 6–11.

Table 6–11 SDO_CRS_COMPOUND View

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system.
COORD_REF_SYS_NAME	VARCHAR2(80)	Name of the coordinate reference system.

Table 6–11 (Cont.) SDO_CRS_COMPOUND View

Column Name	Data Type	Description
CMPD_HORIZ_ SRID	NUMBER(10)	(EPSG-assigned value; not used by Oracle Spatial. The EPSG description is: "For compound CRS only, the code of the horizontal component of the Compound CRS.")
CMPD_VERT_ SRID	NUMBER(10)	(EPSG-assigned value; not used by Oracle Spatial. The EPSG description is: "For compound CRS only, the code of the vertical component of the Compound CRS.")
INFORMATION_ SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).

6.6.13 SDO_CRS_ENGINEERING View

The SDO_CRS_ENGINEERING view contains selected information from the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)) where the COORD_REF_SYS_KIND column value is ENGINEERING. This view contains the columns shown in [Table 6–12](#).

Table 6–12 SDO_CRS_ENGINEERING View

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system.
COORD_REF_SYS_ NAME	VARCHAR2(80)	Name of the coordinate reference system.
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system used for the coordinate reference system. Must match a COORD_SYS_ID value in the SDO_COORD_SYS table (see Section 6.6.11).
DATUM_ID	NUMBER(10)	ID number of the datum used for the coordinate reference system. Must match a DATUM_ID value in the SDO_DATUMS table (see Section 6.6.22). Example: 10115
INFORMATION_ SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).

6.6.14 SDO_CRS_GEOCENTRIC View

The SDO_CRS_GEOCENTRIC view contains selected information from the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)) where the COORD_REF_SYS_KIND column value is GEOCENTRIC. This view contains the columns shown in [Table 6–13](#).

Table 6–13 SDO_CRS_GEOCENTRIC View

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system.
COORD_REF_SYS_ NAME	VARCHAR2(80)	Name of the coordinate reference system.

Table 6–13 (Cont.) SDO_CRS_GEOCENTRIC View

Column Name	Data Type	Description
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system used for the coordinate reference system. Must match a COORD_SYS_ID value in the SDO_COORD_SYS table (see Section 6.6.11).
DATUM_ID	NUMBER(10)	ID number of the datum used for the coordinate reference system. Must match a DATUM_ID value in the SDO_DATUMS table (see Section 6.6.22). Example: 10115
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).

6.6.15 SDO_CRS_GEOGRAPHIC2D View

The SDO_CRS_GEOGRAPHIC2D view contains selected information from the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)) where the COORD_REF_SYS_KIND column value is GEOGRAPHIC2D. This view contains the columns shown in [Table 6–14](#).

Table 6–14 SDO_CRS_GEOGRAPHIC2D View

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system.
COORD_REF_SYS_NAME	VARCHAR2(80)	Name of the coordinate reference system.
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system used for the coordinate reference system. Must match a COORD_SYS_ID value in the SDO_COORD_SYS table (see Section 6.6.11).
DATUM_ID	NUMBER(10)	ID number of the datum used for the coordinate reference system. Must match a DATUM_ID value in the SDO_DATUMS table (see Section 6.6.22). Example: 10115
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).

6.6.16 SDO_CRS_GEOGRAPHIC3D View

The SDO_CRS_GEOGRAPHIC3D view contains selected information from the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)) where the COORD_REF_SYS_KIND column value is GEOGRAPHIC3D. This view contains the columns shown in [Table 6–15](#).

Table 6–15 SDO_CRS_GEOGRAPHIC3D View

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system.
COORD_REF_SYS_NAME	VARCHAR2(80)	Name of the coordinate reference system.

Table 6–15 (Cont.) SDO_CRS_GEOGRAPHIC3D View

Column Name	Data Type	Description
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system used for the coordinate reference system. Must match a COORD_SYS_ID value in the SDO_COORD_SYS table (see Section 6.6.11).
DATUM_ID	NUMBER(10)	ID number of the datum used for the coordinate reference system. Must match a DATUM_ID value in the SDO_DATUMS table (see Section 6.6.22). Example: 10115
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).

6.6.17 SDO_CRS_PROJECTED View

The SDO_CRS_PROJECTED view contains selected information from the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)) where the COORD_REF_SYS_KIND column value is PROJECTED. This view contains the columns shown in [Table 6–16](#).

Table 6–16 SDO_CRS_PROJECTED View

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system.
COORD_REF_SYS_NAME	VARCHAR2(80)	Name of the coordinate reference system.
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system used for the coordinate reference system. Must match a COORD_SYS_ID value in the SDO_COORD_SYS table (see Section 6.6.11).
SOURCE_GEOG_SRID	NUMBER(10)	ID number for the associated geodetic coordinate system.
PROJECTION_CONV_ID	NUMBER(10)	COORD_OP_ID value of the conversion operation used to convert the projected coordinated system to and from the source geographic coordinate system.
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).

6.6.18 SDO_CRS_VERTICAL View

The SDO_CRS_VERTICAL view contains selected information from the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)) where the COORD_REF_SYS_KIND column value is VERTICAL. This view contains the columns shown in [Table 6–17](#).

Table 6–17 SDO_CRS_VERTICAL View

Column Name	Data Type	Description
SRID	NUMBER(10)	ID number of the coordinate reference system.
COORD_REF_SYS_NAME	VARCHAR2(80)	Name of the coordinate reference system.
COORD_SYS_ID	NUMBER(10)	ID number of the coordinate system used for the coordinate reference system. Must match a COORD_SYS_ID value in the SDO_COORD_SYS table (see Section 6.6.11).
DATUM_ID	NUMBER(10)	ID number of the datum used for the coordinate reference system. Must match a DATUM_ID value in the SDO_DATUMS table (see Section 6.6.22). Example: 10115
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition for the coordinate system (Oracle for all rows supplied by Oracle).
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle).

6.6.19 SDO_DATUM_ENGINEERING View

The SDO_DATUM_ENGINEERING view contains selected information from the SDO_DATUMS table (described in [Section 6.6.22](#)) where the DATUM_TYPE column value is ENGINEERING. This view contains the columns shown in [Table 6–18](#).

Table 6–18 SDO_DATUM_ENGINEERING View

Column Name	Data Type	Description
DATUM_ID	NUMBER(10)	ID number of the datum.
DATUM_NAME	VARCHAR2(80)	Name of the datum.
ELLIPSOID_ID	NUMBER(10)	ID number of the ellipsoid used in the datum definition. Must match an ELLIPSOID_ID value in the SDO_ELLIPSOIDS table (see Section 6.6.23). Example: 8045
PRIME_MERIDIAN_ID	NUMBER(10)	ID number of the prime meridian used in the datum definition. Must match a PRIME_MERIDIAN_ID value in the SDO_PRIME_MERIDIANS table (see Section 6.6.26). Example: 8950
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition of the datum. Example: Ordnance Survey of Great Britain.
SHIFT_X	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the x-axis.
SHIFT_Y	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the y-axis.
SHIFT_Z	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the z-axis.
ROTATE_X	NUMBER	Number of arc-seconds of rotation about the x-axis.
ROTATE_Y	NUMBER	Number of arc-seconds of rotation about the y-axis.
ROTATE_Z	NUMBER	Number of arc-seconds of rotation about the z-axis.
SCALE_ADJUST	NUMBER	A value to be used in adjusting the X, Y, and Z values after any shifting and rotation, according to the formula: $1.0 + (\text{SCALE_ADJUST} * 10^{-6})$

6.6.20 SDO_DATUM_GEODETTIC View

The SDO_DATUM_GEODETTIC view contains selected information from the SDO_DATUMS table (described in [Section 6.6.22](#)) where the DATUM_TYPE column value is GEODETTIC. This view contains the columns shown in [Table 6–19](#).

Table 6–19 SDO_DATUM_GEODETTIC View

Column Name	Data Type	Description
DATUM_ID	NUMBER(10)	ID number of the datum.
DATUM_NAME	VARCHAR2(80)	Name of the datum.
ELLIPSOID_ID	NUMBER(10)	ID number of the ellipsoid used in the datum definition. Must match an ELLIPSOID_ID value in the SDO_ELLIPSOIDS table (see Section 6.6.23). Example: 8045
PRIME_MERIDIAN_ID	NUMBER(10)	ID number of the prime meridian used in the datum definition. Must match a PRIME_MERIDIAN_ID value in the SDO_PRIME_MERIDIANS table (see Section 6.6.26). Example: 8950
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition of the datum. Example: Ordnance Survey of Great Britain.
SHIFT_X	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the x-axis.
SHIFT_Y	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the y-axis.
SHIFT_Z	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the z-axis.
ROTATE_X	NUMBER	Number of arc-seconds of rotation about the x-axis.
ROTATE_Y	NUMBER	Number of arc-seconds of rotation about the y-axis.
ROTATE_Z	NUMBER	Number of arc-seconds of rotation about the z-axis.
SCALE_ADJUST	NUMBER	A value to be used in adjusting the X, Y, and Z values after any shifting and rotation, according to the formula: $1.0 + (\text{SCALE_ADJUST} * 10^{-6})$

6.6.21 SDO_DATUM_VERTICAL View

The SDO_DATUM_VERTICAL view contains selected information from the SDO_DATUMS table (described in [Section 6.6.22](#)) where the DATUM_TYPE column value is VERTICAL. This view contains the columns shown in [Table 6–20](#).

Table 6–20 SDO_DATUM_VERTICAL View

Column Name	Data Type	Description
DATUM_ID	NUMBER(10)	ID number of the datum.
DATUM_NAME	VARCHAR2(80)	Name of the datum.
ELLIPSOID_ID	NUMBER(10)	ID number of the ellipsoid used in the datum definition. Must match an ELLIPSOID_ID value in the SDO_ELLIPSOIDS table (see Section 6.6.23). Example: 8045

Table 6–20 (Cont.) SDO_DATUM_VERTICAL View

Column Name	Data Type	Description
PRIME_MERIDIAN_ID	NUMBER(10)	ID number of the prime meridian used in the datum definition. Must match a PRIME_MERIDIAN_ID value in the SDO_PRIME_MERIDIANS table (see Section 6.6.26). Example: 8950
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition of the datum. Example: Ordnance Survey of Great Britain.
SHIFT_X	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the x-axis.
SHIFT_Y	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the y-axis.
SHIFT_Z	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the z-axis.
ROTATE_X	NUMBER	Number of arc-seconds of rotation about the x-axis.
ROTATE_Y	NUMBER	Number of arc-seconds of rotation about the y-axis.
ROTATE_Z	NUMBER	Number of arc-seconds of rotation about the z-axis.
SCALE_ADJUST	NUMBER	A value to be used in adjusting the X, Y, and Z values after any shifting and rotation, according to the formula: $1.0 + (\text{SCALE_ADJUST} * 10^{-6})$

6.6.22 SDO_DATUMS Table

The SDO_DATUMS table contains one row for each datum. This table contains the columns shown in [Table 6–21](#).

Table 6–21 SDO_DATUMS Table

Column Name	Data Type	Description
DATUM_ID	NUMBER(10)	ID number of the datum. Example: 10115
DATUM_NAME	VARCHAR2(80)	Name of the datum. Example: WGS 84
DATUM_TYPE	VARCHAR2(24)	Type of the datum. Example: GEODETIC
ELLIPSOID_ID	NUMBER(10)	ID number of the ellipsoid used in the datum definition. Must match an ELLIPSOID_ID value in the SDO_ELLIPSOIDS table (see Section 6.6.23). Example: 8045
PRIME_MERIDIAN_ID	NUMBER(10)	ID number of the prime meridian used in the datum definition. Must match a PRIME_MERIDIAN_ID value in the SDO_PRIME_MERIDIANS table (see Section 6.6.26). Example: 8950
INFORMATION_SOURCE	VARCHAR2(254)	Provider of the definition of the datum. Example: Ordnance Survey of Great Britain.
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle). Example: EPSG
SHIFT_X	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the x-axis.

Table 6–21 (Cont.) SDO_DATUMS Table

Column Name	Data Type	Description
SHIFT_Y	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the y-axis.
SHIFT_Z	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the z-axis.
ROTATE_X	NUMBER	Number of arc-seconds of rotation about the x-axis.
ROTATE_Y	NUMBER	Number of arc-seconds of rotation about the y-axis.
ROTATE_Z	NUMBER	Number of arc-seconds of rotation about the z-axis.
SCALE_ADJUST	NUMBER	A value to be used in adjusting the X, Y, and Z values after any shifting and rotation, according to the formula: $1.0 + (\text{SCALE_ADJUST} * 10^{-6})$
IS_LEGACY	VARCHAR2(5)	TRUE if the datum definition was included in Oracle Spatial before release 10.2; FALSE if the datum definition is new in Oracle Spatial release 10.2.
LEGACY_CODE	NUMBER(10)	For any EPSG datum that has a semantically identical legacy (in Oracle Spatial before release 10.2) counterpart, the DATUM_ID value of the legacy datum.

See also the information about the following views that are defined based on the value of the DATUM_TYPE column: SDO_DATUM_ENGINEERING (Section 6.6.19), SDO_DATUM_GEODETTIC (Section 6.6.20), and SDO_DATUM_VERTICAL (Section 6.6.21).

6.6.23 SDO_ELLIPSOIDS Table

The SDO_ELLIPSOIDS table contains one row for each ellipsoid. This table contains the columns shown in Table 6–22.

Table 6–22 SDO_ELLIPSOIDS Table

Column Name	Data Type	Description
ELLIPSOID_ID	NUMBER(10)	ID number of the ellipsoid (spheroid). Example: 8045
ELLIPSOID_NAME	VARCHAR2(80)	Name of the ellipsoid. Example: WGS 84
SEMI_MAJOR_AXIS	NUMBER	Radius in meters along the semi-major axis (one-half of the long axis of the ellipsoid).
UOM_ID	NUMBER	ID number of the unit of measurement for the ellipsoid. Matches a value in the UOM_ID column of the SDO_UNITS_OF_MEASURE table (described in Section 6.6.27). Example: 9001
INV_FLATTENING	NUMBER	Inverse flattening of the ellipsoid. That is, $1/f$, where $f = (a-b)/a$, and a is the semi-major axis and b is the semi-minor axis.
SEMI_MINOR_AXIS	NUMBER	Radius in meters along the semi-minor axis (one-half of the short axis of the ellipsoid).
INFORMATION_SOURCE	VARCHAR2(254)	Origin of this information. Example: Kort og Matrikelstyrelsen (KMS), Copenhagen.

Table 6–22 (Cont.) SDO_ELLIPSOIDS Table

Column Name	Data Type	Description
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle). Example: EPSG
IS_LEGACY	VARCHAR2(5)	TRUE if the ellipsoid definition was included in Oracle Spatial before release 10.2; FALSE if the ellipsoid definition is new in Oracle Spatial release 10.2.
LEGACY_CODE	NUMBER	For any EPSG ellipsoid that has a semantically identical legacy (in Oracle Spatial before release 10.2) counterpart, the ELLIPSOID_ID value of the legacy ellipsoid.

6.6.24 SDO_PREFERRED_OPS_SYSTEM Table

The SDO_PREFERRED_OPS_SYSTEM table contains one row for each specification of the user-defined default preferred coordinate transformation operation for a source and target SRID combination. If you insert a row into the SDO_PREFERRED_OPS_SYSTEM table, you are overriding the Oracle default operation for transformations between the specified source and target coordinate systems. The [SDO_CS.CREATE_OBVIOUS_EPSG_RULES](#) procedure inserts many rows into this table. The [SDO_CS.DELETE_ALL_EPSG_RULES](#) procedure deletes all rows from this table if the use_case parameter is null. This table contains the columns shown in [Table 6–23](#).

Table 6–23 SDO_PREFERRED_OPS_SYSTEM Table

Column Name	Data Type	Description
SOURCE_SRID	NUMBER(10)	ID number of the coordinate system (spatial reference system) from which to perform coordinate transformation, using the operation specified by COORD_OP_ID as the default preferred method for transforming to the specified target SRID.
COORD_OP_ID	NUMBER(10)	ID number of the coordinate transformation operation. Matches a value in the COORD_OP_ID column of the SDO_COORD_OPS table (described in Section 6.6.8).
TARGET_SRID	NUMBER(10)	ID number of coordinate system (spatial reference system) into which to perform coordinate transformation using the operation specified by COORD_OP_ID.

6.6.25 SDO_PREFERRED_OPS_USER Table

The SDO_PREFERRED_OPS_USER table contains one row for each specification of a user-defined source and target SRID and coordinate transformation operation. If you insert a row into the SDO_PREFERRED_OPS_USER table, you create a custom transformation between the source and target coordinate systems, and you can specify the name (the USE_CASE column value) of the transformation operation as the use_case parameter value with several SDO_CS functions and procedures. If you specify a use case with the [SDO_CS.DELETE_ALL_EPSG_RULES](#) procedure, rows associated with that use case are deleted from this table. This table contains the columns shown in [Table 6–24](#).

Table 6–24 SDO_PREFERRED_OPS_USER Table

Column Name	Data Type	Description
USE_CASE	VARCHAR2(32)	Name of this specification of a source and target SRID and coordinate transformation operation.
SOURCE_SRID	NUMBER(10)	ID number of the coordinate system (spatial reference system) from which to perform the transformation.
COORD_OP_ID	NUMBER(10)	ID number of the coordinate transformation operation. Matches a value in the COORD_OP_ID column of the SDO_COORD_OPS table (described in Section 6.6.8).
TARGET_SRID	NUMBER(10)	ID number of the coordinate system (spatial reference system) into which to perform the transformation.

6.6.26 SDO_PRIME_MERIDIANS Table

The SDO_PRIME_MERIDIANS table contains one row for each prime meridian that can be used in a datum specification. This table contains the columns shown in [Table 6–25](#).

Table 6–25 SDO_PRIME_MERIDIANS Table

Column Name	Data Type	Description
PRIME_MERIDIAN_ID	NUMBER(10)	ID number of the prime meridian. Example: 8907
PRIME_MERIDIAN_NAME	VARCHAR2(80)	Name of the prime meridian. Example: Bern
GREENWICH_LONGITUDE	FLOAT(49)	Longitude of the prime meridian as an offset from the Greenwich meridian. Example: 7.26225
UOM_ID	NUMBER(10)	ID number of the unit of measurement for the prime meridian. Matches a value in the UOM_ID column of the SDO_UNITS_OF_MEASURE table (described in Section 6.6.27). Example: 9110 for sexagesimal degree
INFORMATION_SOURCE	VARCHAR2(254)	Origin of this information. Example: Bundesamt fur Landestopographie
DATA_SOURCE	VARCHAR2(40)	Organization that supplied the data for this record (if not Oracle). Example: EPSG

6.6.27 SDO_UNITS_OF_MEASURE Table

The SDO_UNITS_OF_MEASURE table contains one row for each unit of measurement. This table contains the columns shown in [Table 6–26](#).

Table 6–26 SDO_UNITS_OF_MEASURE Table

Column Name	Data Type	Description
UOM_ID	NUMBER(10)	ID number of the unit of measurement. Example: 10032
UNIT_OF_MEAS_NAME	VARCHAR2(80)	Name of the unit of measurement. Example: METER
SHORT_NAME	VARCHAR2(80)	Short name (if any) of the unit of measurement.

Table 6–26 (Cont.) SDO_UNITS_OF_MEASURE Table

Column Name	Data Type	Description
UNIT_OF_MEAS_ TYPE	VARCHAR2(80)	Type of measure for which the unit is used: <code>angle</code> for angle unit, <code>area</code> for area unit, or <code>length</code> for distance unit.
TARGET_UOM_ID	NUMBER(10)	ID number of a target unit of measurement. Corresponds to the <code>TARGET_UOM_CODE</code> column in the EPSG Unit of Measure table, which has the following description: "Other UOM of the same type into which the current UOM can be converted using the formula (POSC); POSC factors A and D always equal zero for EPSG supplied units of measure."
FACTOR_B	NUMBER	Corresponds to the <code>FACTOR_B</code> column in the EPSG Unit of Measure table, which has the following description: "A quantity in the target UOM (y) is obtained from a quantity in the current UOM (x) through the conversion: $y = (B/C).x$ "
FACTOR_C	NUMBER	Corresponds to the <code>FACTOR_C</code> column in the EPSG Unit of Measure table.
INFORMATION_ SOURCE	VARCHAR2(254)	Origin of this information. Example: <code>ISO 1000</code> .
DATA_SOURCE	VARCHAR2(40)	Organization providing the data for this record. Example: <code>EPSG</code>
IS_LEGACY	VARCHAR2(5)	TRUE if the unit of measurement definition was included in Oracle Spatial before release 10.2; FALSE if the unit of measurement definition is new in Oracle Spatial release 10.2.
LEGACY_CODE	NUMBER(10)	For any EPSG unit of measure that has a semantically identical legacy (in Oracle Spatial before release 10.2) counterpart, the <code>UOM_ID</code> value of the legacy unit of measure.

6.7 Legacy Tables and Views

In previous releases of Spatial, the coordinate systems functions and procedures used information provided in the following tables, some of which have new names or are now views instead of tables:

- `MDSYS.CS_SRS` (see [Section 6.7.1](#)) defined the valid coordinate systems. It associates each coordinate system with its well-known text description, which is in conformance with the standard published by the Open Geospatial Consortium (<http://www.opengeospatial.org>).
- `MDSYS.SDO_ANGLE_UNITS` (see [Section 6.7.2](#)) defines the valid angle units.
- `MDSYS.SDO_AREA_UNITS` (see [Section 6.7.3](#)) defines the valid area units.
- `MDSYS.SDO_DIST_UNITS` (see [Section 6.7.5](#)) defines the valid distance units.
- `MDSYS.SDO_DATUMS_OLD_FORMAT` and `MDSYS.SDO_DATUMS_OLD_SNAPSHOT` (see [Section 6.7.4](#)) are based on the `MDSYS.SDO_DATUMS` table before release 10.2, which defined valid datums.
- `MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT` and `MDSYS.SDO_ELLIPSOIDS_OLD_SNAPSHOT` (see [Section 6.7.6](#)) are based on the `MDSYS.SDO_ELLIPSOIDS` table before release 10.2, which defined valid ellipsoids.

- `MDSYS.SDO_PROJECTIONS_OLD_FORMAT` and `MDSYS.SDO_PROJECTIONS_OLD_SNAPSHOT` (see [Section 6.7.7](#)) are based on the `MDSYS.SDO_PROJECTIONS` table before release 10.2, which defined the valid map projections.

Note: You should not modify or delete any Oracle-supplied information in these legacy tables.

If you refer to a legacy table in a SQL statement, you must include the `MDSYS.` before the table name.

6.7.1 MDSYS.CS_SRS Table

The `MDSYS.CS_SRS` reference table contains over 4000 rows, one for each valid coordinate system. This table contains the columns shown in [Table 6–27](#).

Table 6–27 *MDSYS.CS_SRS Table*

Column Name	Data Type	Description
<code>CS_NAME</code>	<code>VARCHAR2(68)</code>	A well-known name, often mnemonic, by which a user can refer to the coordinate system.
<code>SRID</code>	<code>NUMBER(38)</code>	The unique ID number (Spatial Reference ID) for a coordinate system. Currently, SRID values 1-999999 are reserved for use by Oracle Spatial, and values 1000000 (1 million) and higher are available for user-defined coordinate systems.
<code>AUTH_SRID</code>	<code>NUMBER(38)</code>	An optional ID number that can be used to indicate how the entry was derived; it might be a foreign key into another coordinate table, for example.
<code>AUTH_NAME</code>	<code>VARCHAR2(256)</code>	An authority name for the coordinate system. Contains <code>Oracle</code> in the supplied table. Users can specify any value in any rows that they add.
<code>WKTEXT</code>	<code>VARCHAR2(2046)</code>	The well-known text (WKT) description of the SRS, as defined by the Open Geospatial Consortium. For more information, see Section 6.7.1.1 .
<code>CS_BOUNDS</code>	<code>SDO_GEOMETRY</code>	An optional <code>SDO_GEOMETRY</code> object that is a polygon with WGS 84 longitude and latitude vertices, representing the spheroidal polygon description of the zone of validity for a projected coordinate system. Must be null for a geographic or non-Earth coordinate system. Is null in all supplied rows.

6.7.1.1 Well-Known Text (WKT)

The `WKTEXT` column of the `MDSYS.CS_SRS` table contains the well-known text (WKT) description of the SRS, as defined by the Open Geospatial Consortium. The following is the WKT EBNF syntax.

```
<coordinate system> ::=
    <horz cs> | <local cs>

<horz cs> ::=
    <geographic cs> | <projected cs>

<projected cs> ::=
    PROJCS [ "<name>", <geographic cs>, <projection>,
            {<parameter>,*} <linear unit> ]
```

```

<projection> ::=
    PROJECTION [ "<name>" ]

<parameter> ::=
    PARAMETER [ "<name>", <number> ]

<geographic cs> ::=
    GEOGCS [ "<name>", <datum>, <prime meridian>, <angular unit> ]

<datum> ::=
    DATUM [ "<name>", <spheroid>
    {, <shift-x>, <shift-y>, <shift-z>
    , <rot-x>, <rot-y>, <rot-z>, <scale_adjust>}
    ]

<spheroid> ::=
    SPHEROID [ "<name>", <semi major axis>, <inverse flattening> ]

<prime meridian> ::=
    PRIMEM [ "<name>", <longitude> ]

<longitude> ::=
    <number>

<semi-major axis> ::=
    <number>

<inverse flattening> ::=
    <number>

<angular unit> ::= <unit>

<linear unit> ::= <unit>

<unit> ::=
    UNIT [ "<name>", <conversion factor> ]

<local cs> ::=
    LOCAL_CS [ "<name>", <local datum>, <linear unit>,
    <axis> {, <axis>}* ]

<local datum> ::=
    LOCAL_DATUM [ "<name>", <datum type>
    {, <shift-x>, <shift-y>, <shift-z>
    , <rot-x>, <rot-y>, <rot-z>, <scale_adjust>}
    ]

<datum type> ::=
    <number>

<axis> ::=
    AXIS [ "<name>", NORTH | SOUTH | EAST |
    WEST | UP | DOWN | OTHER ]

```

Each <parameter> specification is one of the following:

- Standard_Parallel_1 (in decimal degrees)
- Standard_Parallel_2 (in decimal degrees)

- Central_Meridian (in decimal degrees)
- Latitude_of_Origin (in decimal degrees)
- Azimuth (in decimal degrees)
- False_Easting (in the unit of the coordinate system; for example, meters)
- False_Northing (in the unit of the coordinate system; for example, meters)
- Perspective_Point_Height (in the unit of the coordinate system; for example, meters)
- Landsat_Number (must be 1, 2, 3, 4, or 5)
- Path_Number
- Scale_Factor

The default value for each <parameter> specification is 0 (zero). That is, if a specification is needed for a projection but no value is specified in the WKT, Spatial uses a value of 0.

The prime meridian (PRIMEM) is specified in decimal degrees of longitude.

An example of the WKT for a geodetic (geographic) coordinate system is:

```
'GEOGCS [ "Longitude / Latitude (Old Hawaiian)", DATUM ["Old Hawaiian", SPHEROID
["Clarke 1866", 6378206.400000, 294.978698]], PRIMEM [ "Greenwich", 0.000000 ],
UNIT ["Decimal Degree", 0.01745329251994330]]'
```

The WKT definition of the coordinate system is hierarchically nested. The Old Hawaiian geographic coordinate system (GEOGCS) is composed of a named datum (DATUM), a prime meridian (PRIMEM), and a unit definition (UNIT). The datum is in turn composed of a named spheroid and its parameters of semi-major axis and inverse flattening.

An example of the WKT for a projected coordinate system (a Wyoming State Plane) is:

```
'PROJCS["Wyoming 4901, Eastern Zone (1983, meters)", GEOGCS [ "GRS 80", DATUM
["GRS 80", SPHEROID ["GRS 80", 6378137.000000, 298.257222]], PRIMEM [
"Greenwich", 0.000000 ], UNIT ["Decimal Degree", 0.01745329251994330]],
PROJECTION ["Transverse Mercator"], PARAMETER ["Scale_Factor", 0.999938],
PARAMETER ["Central_Meridian", -105.166667], PARAMETER ["Latitude_of_Origin",
40.500000], PARAMETER ["False_Easting", 200000.000000], UNIT ["Meter",
1.000000000000]]'
```

The projected coordinate system contains a nested geographic coordinate system as its basis, as well as parameters that control the projection.

Oracle Spatial supports all common geodetic datums and map projections.

An example of the WKT for a local coordinate system is:

```
LOCAL_CS [ "Non-Earth (Meter)", LOCAL_DATUM ["Local Datum", 0], UNIT ["Meter",
1.0], AXIS ["X", EAST], AXIS["Y", NORTH]]
```

For more information about local coordinate systems, see [Section 6.3](#).

You can use the [SDO_CS.VALIDATE_WKT](#) function, described in [Chapter 13](#), to validate the WKT of any coordinate system defined in the MDSYS.CS_SRS table.

6.7.1.2 Procedures for Updating the Well-Known Text

If you insert or delete a row in the SDO_COORD_REF_SYSTEM view (described in [Section 6.6.10](#)), Spatial automatically updates the WKTEXT column in the MDSYS.CS_

SRS table. (The format of the WKTEXT column is described in [Section 6.7.1.1](#).) However, if you update an existing row in the SDO_COORD_REF_SYSTEM view, the well-known text (WKT) value is not automatically updated.

In addition, information relating to coordinate reference systems is also stored in several other system tables, including SDO_DATUMS (described in [Section 6.6.22](#)), SDO_ELLIPSOIDS (described in [Section 6.6.23](#)), and SDO_PRIME_MERIDIANS (described in [Section 6.6.26](#)). If you add, delete, or modify information in these tables, the WKTEXT values in the MDSYS.CS_SRS table are not automatically updated. For example, if you update an ellipsoid flattening value in the SDO_ELLIPSOIDS table, the well-known text string for the associated coordinate system is not updated.

However, you can manually update the WKTEXT values in the in the MDSYS.CS_SRS table by using any of several procedures whose names start with *UPDATE_WKTS_FOR* (for example, [SDO_CS.UPDATE_WKTS_FOR_ALL_EPSG_CRS](#) and [SDO_CS.UPDATE_WKTS_FOR_EPSG_DATUM](#)). If the display of SERVEROUTPUT information is enabled, these procedures display a message identifying the SRID value for each row in the MDSYS.CS_SRS table whose WKTEXT value is being updated. These procedures are described in [Chapter 13](#).

6.7.2 MDSYS.SDO_ANGLE_UNITS View

The MDSYS.SDO_ANGLE_UNITS reference view contains one row for each valid angle UNIT specification in the well-known text (WKT) description in the coordinate system definition. The WKT is described in [Section 6.7.1.1](#).

The MDSYS.SDO_ANGLE_UNITS view is based on the SDO_UNITS_OF MEASURE table (described in [Section 6.6.27](#)), and it contains the columns shown in [Table 6–28](#).

Table 6–28 MDSYS.SDO_ANGLE_UNITS View

Column Name	Data Type	Description
SDO_UNIT	VARCHAR2(32)	Name of the angle unit (often a shortened form of the UNIT_NAME value). Use the SDO_UNIT value with the <i>from_unit</i> and <i>to_unit</i> parameters of the SDO_UTIL.CONVERT_UNIT function.
UNIT_NAME	VARCHAR2(100)	Name of the angle unit. Specify a value from this column in the UNIT specification of the WKT for any user-defined coordinate system. Examples: Decimal Degree, Radian, Decimal Second, Decimal Minute, Gon, Grad.
CONVERSION_FACTOR	NUMBER	The ratio of the specified unit to one radian. For example, the ratio of Decimal Degree to Radian is 0.017453293.

6.7.3 MDSYS.SDO_AREA_UNITS View

The MDSYS.SDO_AREA_UNITS reference view contains one row for each valid area UNIT specification in the well-known text (WKT) description in the coordinate system definition. The WKT is described in [Section 6.7.1.1](#).

The MDSYS.SDO_AREA_UNITS view is based on the SDO_UNITS_OF MEASURE table (described in [Section 6.6.27](#)), and it contains the columns shown in [Table 6–29](#).

Table 6–29 SDO_AREA_UNITS View

Column Name	Data Type	Purpose
SDO_UNIT	VARCHAR2	Values are taken from the SHORT_NAME column of the SDO_UNITS_OF_MEASURE table.
UNIT_NAME	VARCHAR2	Values are taken from the UNIT_OF_MEAS_NAME column of the SDO_UNITS_OF_MEASURE table.
CONVERSION_FACTOR	NUMBER	Ratio of the unit to 1 square meter. For example, the conversion factor for a square meter is 1.0, and the conversion factor for a square mile is 2589988.

6.7.4 MDSYS.SDO_DATUMS_OLD_FORMAT and SDO_DATUMS_OLD_SNAPSHOT Tables

The MDSYS.SDO_DATUMS_OLD_FORMAT and MDSYS.SDO_DATUMS_OLD_SNAPSHOT reference tables contain one row for each valid DATUM specification in the well-known text (WKT) description in the coordinate system definition. (The WKT is described in [Section 6.7.1.1](#).)

- MDSYS.SDO_DATUMS_OLD_FORMAT contains the new data in the old format (that is, EPSG-based datum specifications in a table using the format from before release 10.2).
- MDSYS.SDO_DATUMS_OLD_SNAPSHOT contains the old data in the old format (that is, datum specifications and table format from before release 10.2).

These tables contain the columns shown in [Table 6–30](#).

Table 6–30 MDSYS.SDO_DATUMS_OLD_FORMAT and SDO_DATUMS_OLD_SNAPSHOT Tables

Column Name	Data Type	Description
NAME	VARCHAR2(80) for OLD_FORMAT VARCHAR2(64) for OLD_SNAPSHOT	Name of the datum. Specify a value (Oracle-supplied or user-defined) from this column in the DATUM specification of the WKT for any user-defined coordinate system. Examples: Adindan, Afgooye, Ain el Abd 1970, Anna 1 Astro 1965, Arc 1950, Arc 1960, Ascension Island 1958.
SHIFT_X	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the x-axis.
SHIFT_Y	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the y-axis.
SHIFT_Z	NUMBER	Number of meters to shift the ellipsoid center relative to the center of the WGS 84 ellipsoid on the z-axis.
ROTATE_X	NUMBER	Number of arc-seconds of rotation about the x-axis.
ROTATE_Y	NUMBER	Number of arc-seconds of rotation about the y-axis.
ROTATE_Z	NUMBER	Number of arc-seconds of rotation about the z-axis.
SCALE_ADJUST	NUMBER	A value to be used in adjusting the X, Y, and Z values after any shifting and rotation, according to the formula: $1.0 + (\text{SCALE_ADJUST} * 10^{-6})$

The following are the names (in tabular format) of the datums in these tables:

Adindan	Afgooye	Ain el Abd 1970
Anna 1 Astro 1965	Arc 1950	Arc 1960
Ascension Island 1958	Astro B4 Sorol Atoll	Astro Beacon E
Astro DOS 71/4	Astronomic Station 1952	Australian Geodetic 1966
Australian Geodetic 1984	Belgium Hayford	Bellevue (IGN)
Bermuda 1957	Bogota Observatory	CH 1903 (Switzerland)
Campo Inchauspe	Canton Astro 1966	Cape
Cape Canaveral	Carthage	Chatham 1971
Chua Astro	Corrego Alegre	DHDN (Potsdam/Rauenberg)
DOS 1968	Djakarta (Batavia)	Easter Island 1967
European 1950	European 1979	European 1987
GRS 67	GRS 80	GUX 1 Astro
Gandajika Base	Geodetic Datum 1949	Guam 1963
Hito XVIII 1963	Hjorsey 1955	Hong Kong 1963
Hu-Tzu-Shan	ISTS 073 Astro 1969	Indian (Bangladesh, etc.)
Indian (Thailand/Vietnam)	Ireland 1965	Johnston Island 1961
Kandawala	Kerguelen Island	Kertau 1948
L.C. 5 Astro	Liberia 1964	Lisboa (DLx)
Luzon (Mindanao Island)	Luzon (Philippines)	Mahe 1971
Marco Astro	Massawa	Melrica 1973 (D73)
Merchich	Midway Astro 1961	Minna
NAD 27 (Alaska)	NAD 27 (Bahamas)	NAD 27 (Canada)
NAD 27 (Canal Zone)	NAD 27 (Caribbean)	NAD 27 (Central America)
NAD 27 (Continental US)	NAD 27 (Cuba)	NAD 27 (Greenland)
NAD 27 (Mexico)	NAD 27 (Michigan)	NAD 27 (San Salvador)
NAD 83	NTF (Greenwich meridian)	NTF (Paris meridian)
NWGL 10	Nahrwan (Masirah Island)	Nahrwan (Saudi Arabia)
Nahrwan (Un. Arab Emirates)	Naparima, BWI	Netherlands Bessel
Observatorio 1966	Old Egyptian	Old Hawaiian
Oman	Ordinance Survey Great Brit	Pico de las Nieves
Pitcairn Astro 1967	Provisional South American	Puerto Rico
Pulkovo 1942	Qatar National	Qornoq

RT 90 (Sweden)	Reunion	Rome 1940
Santo (DOS)	Sao Braz	Sapper Hill 1943
Schwarzeck	South American 1969	South Asia
Southeast Base	Southwest Base	Timbalai 1948
Tokyo	Tristan Astro 1968	Viti Levu 1916
WGS 60	WGS 66	WGS 72
WGS 84	Wake-Eniwetok 1960	Yacare
Zanderij		

6.7.5 MDSYS.SDO_DIST_UNITS View

The MDSYS.SDO_DIST_UNITS reference view contains one row for each valid distance UNIT specification in the well-known text (WKT) description in the coordinate system definition. The WKT is described in [Section 6.7.1.1](#).

The MDSYS.SDO_DIST_UNITS view is based on the SDO_UNITS_OF MEASURE table (described in [Section 6.6.27](#)), and it contains the columns shown in [Table 6–31](#).

Table 6–31 MDSYS.SDO_DIST_UNITS View

Column Name	Data Type	Description
SDO_UNIT	VARCHAR2	Values are taken from the SHORT_NAME column of the SDO_UNITS_OF MEASURE table.
UNIT_NAME	VARCHAR2	Values are taken from the UNIT_OF_MEAS_NAME column of the SDO_UNITS_OF MEASURE table.
CONVERSION_FACTOR	NUMBER	Ratio of the unit to 1 meter. For example, the conversion factor for a meter is 1.0, and the conversion factor for a mile is 1609.344.

6.7.6 MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT and SDO_ELLIPSOIDS_OLD_SNAPSHOT Tables

The MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT and MDSYS.SDO_ELLIPSOIDS_OLD_SNAPSHOT reference tables contain one row for each valid SPHEROID specification in the well-known text (WKT) description in the coordinate system definition. (The WKT is described in [Section 6.7.1.1](#).)

- MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT contains the new data in the old format (that is, EPSG-based ellipsoid specifications in a table using the format from before release 10.2).
- MDSYS.SDO_ELLIPSOIDS_OLD_SNAPSHOT contains the old data in the old format (that is, ellipsoid specifications and table format from before release 10.2).

These tables contain the columns shown in [Table 6–32](#).

Table 6–32 MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT and SDO_ELLIPSOIDS_OLD_SNAPSHOT Tables

Column Name	Data Type	Description
NAME	VARCHAR2(80) for OLD_FORMAT VARCHAR2(64) for OLD_SNAPSHOT	Name of the ellipsoid (spheroid). Specify a value from this column in the SPHEROID specification of the WKT for any user-defined coordinate system. Examples: Clarke 1866, WGS 72, Australian, Krassovsky, International 1924.
SEMI_MAJOR_AXIS	NUMBER	Radius in meters along the semi-major axis (one-half of the long axis of the ellipsoid).
INVERSE_FLATTENING	NUMBER	Inverse flattening of the ellipsoid. That is, $1/f$, where $f = (a-b)/a$, and a is the semi-major axis and b is the semi-minor axis.

The following are the names (in tabular format) of the ellipsoids in these tables:

Airy 1830	Airy 1830 (Ireland 1965)	Australian
Bessel 1841	Bessel 1841 (NGO 1948)	Bessel 1841 (Schwarzeck)
Clarke 1858	Clarke 1866	Clarke 1866 (Michigan)
Clarke 1880	Clarke 1880 (Arc 1950)	Clarke 1880 (IGN)
Clarke 1880 (Jamaica)	Clarke 1880 (Merchich)	Clarke 1880 (Palestine)
Everest	Everest (Kalianpur)	Everest (Kertau)
Everest (Timbalai)	Fischer 1960 (Mercury)	Fischer 1960 (South Asia)
Fischer 1968	GRS 67	GRS 80
Hayford	Helmert 1906	Hough
IAG 75	Indonesian	International 1924
Krassovsky	MERIT 83	NWL 10D
NWL 9D	New International 1967	OSU86F
OSU91A	Plessis 1817	South American 1969
Sphere (6370997m)	Struve 1860	WGS 60
WGS 66	WGS 72	WGS 84
Walbeck	War Office	

6.7.7 MDSYS.SDO_PROJECTIONS_OLD_FORMAT and SDO_PROJECTIONS_OLD_SNAPSHOT Tables

The MDSYS.SDO_PROJECTIONS_OLD_FORMAT and MDSYS.SDO_PROJECTIONS_OLD_SNAPSHOT reference tables contain one row for each valid PROJECTION specification in the well-known text (WKT) description in the coordinate system definition. (The WKT is described in [Section 6.7.1.1](#).)

- MDSYS.SDO_PROJECTIONS_OLD_FORMAT contains the new data in the old format (that is, EPSG-based projection specifications in a table using the format from before release 10.2).
- MDSYS.SDO_PROJECTIONS_OLD_SNAPSHOT contains the old data in the old format (that is, projection specifications and table format from before release 10.2).

These tables contains the column shown in [Table 6–33](#).

Table 6–33 MDSYS.SDO_PROJECTIONS_OLD_FORMAT and SDO_PROJECTIONS_OLD_SNAPSHOT Tables

Column Name	Data Type	Description
NAME	VARCHAR2(80) for OLD_FORMAT VARCHAR2(64) for OLD_SNAPSHOT	Name of the map projection. Specify a value from this column in the PROJECTION specification of the WKT for any user-defined coordinate system. Examples: Geographic (Lat/Long), Universal Transverse Mercator, State Plane Coordinates, Albers Conical Equal Area.

The following are the names (in tabular format) of the projections in these tables:

Alaska Conformal	Albers Conical Equal Area
Azimuthal Equidistant	Bonne
Cassini	Cylindrical Equal Area
Eckert IV	Eckert VI
Equidistant Conic	Equirectangular
Gall	General Vertical Near-Side Perspective
Geographic (Lat/Long)	Gnomonic
Hammer	Hotine Oblique Mercator
Interrupted Goode Homolosine	Interrupted Mollweide
Lambert Azimuthal Equal Area	Lambert Conformal Conic
Lambert Conformal Conic (Belgium 1972)	Mercator
Miller Cylindrical	Mollweide
New Zealand Map Grid	Oblated Equal Area
Orthographic	Polar Stereographic
Polyconic	Robinson
Sinusoidal	Space Oblique Mercator
State Plane Coordinates	Stereographic
Swiss Oblique Mercator	Transverse Mercator
Transverse Mercator Danish System 34 Jylland-Fyn	Transverse Mercator Danish System 45 Bornholm
Transverse Mercator Finnish KKJ	Transverse Mercator Sjaelland
Universal Transverse Mercator	Van der Grinten
Wagner IV	Wagner VII

6.8 Creating a User-Defined Coordinate Reference System

If the coordinate systems supplied by Oracle are not sufficient for your needs, you can create user-defined coordinate reference systems.

Note: As mentioned in [Section 6.1.1](#), the terms *coordinate system* and *coordinate reference system* (CRS) are often used interchangeably, although coordinate reference systems must be Earth-based.

The exact steps for creating a user-defined CRS depend on whether it is geodetic or projected. In both cases, supply information about the coordinate system (coordinate axes, axis names, unit of measurement, and so on). For a geodetic CRS, supply information about the datum (ellipsoid, prime meridian, and so on), as explained in [Section 6.8.1](#). For a projected CRS, supply information about the source (geodetic) CRS and the projection (operation and parameters), as explained in [Section 6.8.2](#).

For any user-defined coordinate system, the SRID value should be 1000000 (1 million) or higher.

6.8.1 Creating a Geodetic CRS

If the necessary unit of measurement, coordinate axes, SDO_COORD_SYS table row, ellipsoid, prime meridian, and datum are already defined, insert a row into the SDO_COORD_REF_SYSTEM view (described in [Section 6.6.10](#)) to define the new geodetic CRS.

[Example 6-2](#) inserts the definition for a hypothetical geodetic CRS named My Own NAD27 (which, except for its SRID and name, is the same as the NAD27 CRS supplied by Oracle).

Example 6-2 Creating a User-Defined Geodetic Coordinate Reference System

```
INSERT INTO SDO_COORD_REF_SYSTEM (
    SRID,
    COORD_REF_SYS_NAME,
    COORD_REF_SYS_KIND,
    COORD_SYS_ID,
    DATUM_ID,
    GEOG_CRD_DATUM_ID,
    SOURCE_GEOG_SRID,
    PROJECTION_CONV_ID,
    CMPD_HORIZ_SRID,
    CMPD_VERT_SRID,
    INFORMATION_SOURCE,
    DATA_SOURCE,
    IS_LEGACY,
    LEGACY_CODE,
    LEGACY_WKTEXT,
    LEGACY_CS_BOUNDS,
    IS_VALID,
    SUPPORTS_SDO_GEOMETRY)
VALUES (
    9994267,
    'My Own NAD27',
    'GEOGRAPHIC2D',
    6422,
    6267,
    6267,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
```

```
'EPSG',
'FALSE',
NULL,
NULL,
NULL,
'TRUE',
'TRUE');
```

If the necessary information for the definition does not already exist, follow these steps, as needed, to define the information before you insert the row into the SDO_COORD_REF_SYSTEM view:

1. If the unit of measurement is not already defined in the SDO_UNITS_OF_MEASURE table (described in [Section 6.6.27](#)), insert a row into that table to define the new unit of measurement.
2. If the coordinate axes are not already defined in the SDO_COORD_AXES table (described in [Section 6.6.1](#)), insert one row into that table for each new coordinate axis.
3. If an appropriate entry for the coordinate system does not already exist in the SDO_COORD_SYS table (described in [Section 6.6.11](#)), insert a row into that table. [Example 6-3](#) inserts the definition for a fictitious coordinate system.

Example 6-3 Inserting a Row into the SDO_COORD_SYS Table

```
INSERT INTO SDO_COORD_SYS (
    COORD_SYS_ID,
    COORD_SYS_NAME,
    COORD_SYS_TYPE,
    DIMENSION,
    INFORMATION_SOURCE,
    DATA_SOURCE)
VALUES (
    9876543,
    'My custom CS. Axes: lat, long. Orientations: north, east. UoM: deg',
    'ellipsoidal',
    2,
    'Myself',
    'Myself');
```

4. If the ellipsoid is not already defined in the SDO_ELLIPSOIDS table (described in [Section 6.6.23](#)), insert a row into that table to define the new ellipsoid.
5. If the prime meridian is not already defined in the SDO_PRIME_MERIDIANS table (described in [Section 6.6.26](#)), insert a row into that table to define the new prime meridian.
6. If the datum is not already defined in the SDO_DATUMS table (described in [Section 6.6.22](#)), insert a row into that table to define the new datum.

6.8.2 Creating a Projected CRS

If the necessary unit of measurement, coordinate axes, SDO_COORD_SYS table row, source coordinate system, projection operation, and projection parameters are already defined, insert a row into the SDO_COORD_REF_SYSTEM view (described in [Section 6.6.10](#)) to define the new projected CRS.

[Example 6-4](#) inserts the definition for a hypothetical projected CRS named My Own NAD27 / Cuba Norte (which, except for its SRID and name, is the same as the NAD27 / Cuba Norte CRS supplied by Oracle).

Example 6–4 Creating a User-Defined Projected Coordinate Reference System

```

INSERT INTO SDO_COORD_REF_SYSTEM (
    SRID,
    COORD_REF_SYS_NAME,
    COORD_REF_SYS_KIND,
    COORD_SYS_ID,
    DATUM_ID,
    GEOG_CRS_DATUM_ID,
    SOURCE_GEOG_SRID,
    PROJECTION_CONV_ID,
    CMPD_HORIZ_SRID,
    CMPD_VERT_SRID,
    INFORMATION_SOURCE,
    DATA_SOURCE,
    IS_LEGACY,
    LEGACY_CODE,
    LEGACY_WKTEXT,
    LEGACY_CS_BOUNDS,
    IS_VALID,
    SUPPORTS_SDO_GEOMETRY)
VALUES (
    9992085,
    'My Own NAD27 / Cuba Norte',
    'PROJECTED',
    4532,
    NULL,
    6267,
    4267,
    18061,
    NULL,
    NULL,
    'Institut Cubano di Hidrografia (ICH)',
    'EPSG',
    'FALSE',
    NULL,
    NULL,
    NULL,
    'TRUE',
    'TRUE');

```

If the necessary information for the definition does not already exist, follow these steps, as needed, to define the information before you insert the row into the `SDO_COORD_REF_SYSTEM` view:

1. If the unit of measurement is not already defined in the `SDO_UNITS_OF_MEASURE` table (described in [Section 6.6.27](#)), insert a row into that table to define the new unit of measurement.
2. If the coordinate axes are not already defined in the `SDO_COORD_AXES` table (described in [Section 6.6.1](#)), insert one row into that table for each new coordinate axis.
3. If an appropriate entry for the coordinate system does not already exist in `SDO_COORD_SYS` table (described in [Section 6.6.11](#)), insert a row into that table. (See [Example 6–3](#) in [Section 6.8.1](#)).
4. If the projection operation is not already defined in the `SDO_COORD_OPS` table (described in [Section 6.6.8](#)), insert a row into that table to define the new projection operation. [Example 6–5](#) shows the statement used to insert information about coordinate operation ID 18061, which is supplied by Oracle.

Example 6–5 Inserting a Row into the SDO_COORD_OPS Table

```

INSERT INTO SDO_COORD_OPS (
    COORD_OP_ID,
    COORD_OP_NAME,
    COORD_OP_TYPE,
    SOURCE_SRID,
    TARGET_SRID,
    COORD_TFM_VERSION,
    COORD_OP_VARIANT,
    COORD_OP_METHOD_ID,
    UOM_ID_SOURCE_OFFSETS,
    UOM_ID_TARGET_OFFSETS,
    INFORMATION_SOURCE,
    DATA_SOURCE,
    SHOW_OPERATION,
    IS_LEGACY,
    LEGACY_CODE,
    REVERSE_OP,
    IS_IMPLEMENTED_FORWARD,
    IS_IMPLEMENTED_REVERSE)
VALUES (
    18061,
    'Cuba Norte',
    'CONVERSION',
    NULL,
    NULL,
    NULL,
    NULL,
    9801,
    NULL,
    NULL,
    NULL,
    'EPSG',
    1,
    'FALSE',
    NULL,
    1,
    1,
    1);

```

5. If the parameters for the projection operation are not already defined in the SDO_COORD_OP_PARAM_VALS table (described in [Section 6.6.5](#)), insert one row into that table for each new parameter. [Example 6–6](#) shows the statement used to insert information about parameters with ID values 8801, 8802, 8805, 8806, and 8807, which are supplied by Oracle.

Example 6–6 Inserting a Row into the SDO_COORD_OP_PARAM_VALS Table

```

INSERT INTO SDO_COORD_OP_PARAM_VALS (
    COORD_OP_ID,
    COORD_OP_METHOD_ID,
    PARAMETER_ID,
    PARAMETER_VALUE,
    PARAM_VALUE_FILE_REF,
    UOM_ID)
VALUES (
    18061,
    9801,
    8801,
    22.21,

```

```
        NULL,
        9110);

INSERT INTO SDO_COORD_OP_PARAM_VALS (
    COORD_OP_ID,
    COORD_OP_METHOD_ID,
    PARAMETER_ID,
    PARAMETER_VALUE,
    PARAM_VALUE_FILE_REF,
    UOM_ID)
VALUES (
    18061,
    9801,
    8802,
    -81,
    NULL,
    9110);

INSERT INTO SDO_COORD_OP_PARAM_VALS (
    COORD_OP_ID,
    COORD_OP_METHOD_ID,
    PARAMETER_ID,
    PARAMETER_VALUE,
    PARAM_VALUE_FILE_REF,
    UOM_ID)
VALUES (
    18061,
    9801,
    8805,
    .99993602,
    NULL,
    9201);

INSERT INTO SDO_COORD_OP_PARAM_VALS (
    COORD_OP_ID,
    COORD_OP_METHOD_ID,
    PARAMETER_ID,
    PARAMETER_VALUE,
    PARAM_VALUE_FILE_REF,
    UOM_ID)
VALUES (
    18061,
    9801,
    8806,
    500000,
    NULL,
    9001);

INSERT INTO SDO_COORD_OP_PARAM_VALS (
    COORD_OP_ID,
    COORD_OP_METHOD_ID,
    PARAMETER_ID,
    PARAMETER_VALUE,
    PARAM_VALUE_FILE_REF,
    UOM_ID)
VALUES (
    18061,
    9801,
    8807,
    280296.016,
```

```
NULL,  
9001);
```

6.9 Notes and Restrictions with Coordinate Systems Support

The following notes and restrictions apply to coordinate systems support in the current release of Oracle Spatial.

If you have geodetic data, see [Section 6.2](#) for additional considerations, guidelines, and restrictions.

6.9.1 Different Coordinate Systems for Geometries with Operators and Functions

For Spatial operators (described in [Chapter 11](#)) that take two geometries as input parameters, if the geometries are based on different coordinate systems, the query window (the second geometry) is transformed to the coordinate system of the first geometry before the operation is performed. This transformation is a temporary internal operation performed by Spatial; it does not affect any stored query-window geometry.

For SDO_GEOM package geometry functions (described in [Chapter 15](#)) that take two geometries as input parameters, both geometries must be based on the same coordinate system.

6.9.2 3D LRS Functions Not Supported with Geodetic Data

In the current release, the 3D formats of LRS functions (explained in [Section 7.4](#)) are not supported with geodetic data.

6.9.3 Functions Supported by Approximations with Geodetic Data

In the current release, the following functions are supported by approximations with geodetic data:

- [SDO_GEOM.SDO_BUFFER](#)
- [SDO_GEOM.SDO_CENTROID](#)
- [SDO_GEOM.SDO_CONVEXHULL](#)

When these functions are used on data with geodetic coordinates, they internally perform the operations in an implicitly generated local-tangent-plane Cartesian coordinate system and then transform the results to the geodetic coordinate system. For [SDO_GEOM.SDO_BUFFER](#), generated arcs are approximated by line segments before the back-transform.

6.9.4 Unknown CRS and NaC Coordinate Reference Systems

The following coordinate reference systems are provided for Oracle internal use and for other possible special uses:

- unknown CRS (SRID 999999) means that the coordinate system is unknown, and its space could be geodetic or Cartesian. Contrast this with specifying a null coordinate reference system, which indicates an unknown coordinate system with a Cartesian space.
- NaC (SRID 999998) means *Not-a-CRS*. Its name is patterned after the NaN (*Not-a-Number*) value in Java. It is intended for potential use with nonspatial geometries.

The following restrictions apply to geometries based on the unknown CRS and NaC coordinate reference systems:

- You cannot perform coordinate system transformations on these geometries.
- Operations that require a coordinate system will return a null value when performed on these geometries. These operations include finding the area or perimeter of a geometry, creating a buffer, densifying an arc, and computing the aggregate centroid.

6.10 U.S. National Grid Support

The U.S. National Grid is a point coordinate representation using a single alphanumeric coordinate (for example, 18SUJ2348316806479498). This approach contrasts with the use of numeric coordinates to represent the location of a point, as is done with Oracle Spatial and EPSG. A good description of the U.S. National Grid is available at <http://www.ngs.noaa.gov/TOOLS/usng.html>.

To support the U.S. National Grid in Spatial, the SDO_GEOMETRY type cannot be used because it is based on numeric coordinates. Instead, a point in U.S. National Grid format is represented as a single string of type VARCHAR2. To allow conversion between the SDO_GEOMETRY format and the U.S. National grid format, the SDO_CS package (documented in [Chapter 13](#)) contains the following functions:

- [SDO_CS.FROM_USNG](#)
- [SDO_CS.TO_USNG](#)

6.11 Example of Coordinate System Transformation

This section presents a simplified example that uses coordinate system transformation functions and procedures. It refers to concepts that are explained in this chapter and uses functions documented in [Chapter 13](#).

[Example 6–7](#) uses mostly the same geometry data (cola markets) as in [Section 2.1](#), except that instead of null SDO_SRID values, the SDO_SRID value 8307 is used. That is, the geometries are defined as using the coordinate system whose SRID is 8307 and whose well-known name is "Longitude / Latitude (WGS 84)". This is probably the most widely used coordinate system, and it is the one used for global positioning system (GPS) devices. The geometries are then transformed using the coordinate system whose SRID is 8199 and whose well-known name is "Longitude / Latitude (Arc 1950)".

[Example 6–7](#) uses the geometries illustrated in [Figure 2–1](#) in [Section 2.1](#), except that `cola_d` is a rectangle (here, a square) instead of a circle, because arcs are not supported with geodetic coordinate systems.

[Example 6–7](#) does the following:

- Creates a table (COLA_MARKETS_CS) to hold the spatial data
- Inserts rows for four areas of interest (`cola_a`, `cola_b`, `cola_c`, `cola_d`), using the SDO_SRID value 8307
- Updates the USER_SDO_GEOM_METADATA view to reflect the dimension of the areas, using the SDO_SRID value 8307
- Creates a spatial index (COLA_SPATIAL_IDX_CS)
- Performs some transformation operations (single geometry and entire layer)

[Example 6–8](#) includes the output of the SELECT statements in [Example 6–7](#).

Example 6-7 Simplified Example of Coordinate System Transformation

```

-- Create a table for cola (soft drink) markets in a
-- given geography (such as city or state).

CREATE TABLE cola_markets_cs (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape SDO_GEOMETRY);

-- The next INSERT statement creates an area of interest for
-- Cola A. This area happens to be a rectangle.
-- The area could represent any user-defined criterion: for
-- example, where Cola A is the preferred drink, where
-- Cola A is under competitive pressure, where Cola A
-- has strong growth potential, and so on.

INSERT INTO cola_markets_cs VALUES(
  1,
  'cola_a',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    8307, -- SRID for 'Longitude / Latitude (WGS 84)' coordinate system
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), -- polygon
    SDO_ORDINATE_ARRAY(1,1, 5,1, 5,7, 1,7, 1,1) -- All vertices must
    -- be defined for rectangle with geodetic data.
  )
);

-- The next two INSERT statements create areas of interest for
-- Cola B and Cola C. These areas are simple polygons (but not
-- rectangles).

INSERT INTO cola_markets_cs VALUES(
  2,
  'cola_b',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    8307,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), -- one polygon (exterior polygon ring)
    SDO_ORDINATE_ARRAY(5,1, 8,1, 8,6, 5,7, 5,1)
  )
);

INSERT INTO cola_markets_cs VALUES(
  3,
  'cola_c',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    8307,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), --one polygon (exterior polygon ring)
    SDO_ORDINATE_ARRAY(3,3, 6,3, 6,5, 4,5, 3,3)
  )
);

-- Insert a rectangle (here, square) instead of a circle as in the original,
-- because arcs are not supported with geodetic coordinate systems.
INSERT INTO cola_markets_cs VALUES(

```

```

4,
'cola_d',
SDO_GEOMETRY(
  2003, -- two-dimensional polygon
  8307, -- SRID for 'Longitude / Latitude (WGS 84)' coordinate system
  NULL,
  SDO_ELEM_INFO_ARRAY(1,1003,1), -- polygon
  SDO_ORDINATE_ARRAY(10,9, 11,9, 11,10, 10,10, 10,9) -- All vertices must
  -- be defined for rectangle with geodetic data.
)
);

-----
-- UPDATE METADATA VIEW --
-----

-- Update the USER_SDO_GEOM_METADATA view. This is required
-- before the Spatial index can be created. Do this only once for each
-- layer (table-column combination; here: cola_markets_cs and shape).

INSERT INTO user_sdo_geom_metadata
  (TABLE_NAME,
   COLUMN_NAME,
   DIMINFO,
   SRID)
VALUES (
  'cola_markets_cs',
  'shape',
  SDO_DIM_ARRAY(
    SDO_DIM_ELEMENT('Longitude', -180, 180, 10), -- 10 meters tolerance
    SDO_DIM_ELEMENT('Latitude', -90, 90, 10) -- 10 meters tolerance
  ),
  8307 -- SRID for 'Longitude / Latitude (WGS 84)' coordinate system
);

-----
-- CREATE THE SPATIAL INDEX --
-----

CREATE INDEX cola_spatial_idx_cs
ON cola_markets_cs(shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;

-----
-- TEST COORDINATE SYSTEM TRANSFORMATION --
-----

-- Return the transformation of cola_c using to_srid 8199
-- ('Longitude / Latitude (Arc 1950)')
SELECT c.name, SDO_CS.TRANSFORM(c.shape, m.diminfo, 8199)
FROM cola_markets_cs c, user_sdo_geom_metadata m
WHERE m.table_name = 'COLA_MARKETS_CS' AND m.column_name = 'SHAPE'
AND c.name = 'cola_c';

-- Same as preceding, but using to_sname parameter.
SELECT c.name, SDO_CS.TRANSFORM(c.shape, m.diminfo, 'Longitude / Latitude (Arc
1950)')
FROM cola_markets_cs c, user_sdo_geom_metadata m
WHERE m.table_name = 'COLA_MARKETS_CS' AND m.column_name = 'SHAPE'
AND c.name = 'cola_c';

-- Transform the entire SHAPE layer and put results in the table

```

```
-- named cola_markets_cs_8199, which the procedure will create.
CALL SDO_CS.TRANSFORM_LAYER('COLA_MARKETS_CS','SHAPE','COLA_MARKETS_CS_
8199',8199);

-- Select all from the old (existing) table.
SELECT * from cola_markets_cs;

-- Select all from the new (layer transformed) table.
SELECT * from cola_markets_cs_8199;

-- Show metadata for the new (layer transformed) table.
DESCRIBE cola_markets_cs_8199;

-- Use a geodetic MBR with SDO_FILTER.
SELECT c.name FROM cola_markets_cs c WHERE
  SDO_FILTER(c.shape,
    SDO_GEOMETRY(
      2003,
      8307,    -- SRID for WGS 84 longitude/latitude
      NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(6,5, 10,10))
    ) = 'TRUE';
```

Example 6–8 shows the output of the SELECT statements in **Example 6–7**. Notice the slight differences between the coordinates in the original geometries (SRID 8307) and the transformed coordinates (SRID 8199) -- for example, (1, 1, 5, 1, 5, 7, 1, 7, 1, 1) and (1.00078604, 1.00274579, 5.00069354, 1.00274488, 5.0006986, 7.00323528, 1.00079179, 7.00324162, 1.00078604, 1.00274579) for cola_a.

Example 6–8 Output of SELECT Statements in Coordinate System Transformation Example

```
SQL> -- Return the transformation of cola_c using to_srid 8199
SQL> -- ('Longitude / Latitude (Arc 1950)')
SQL> SELECT c.name, SDO_CS.TRANSFORM(c.shape, m.diminfo, 8199)
  2   FROM cola_markets_cs c, user_sdo_geom_metadata m
  3   WHERE m.table_name = 'COLA_MARKETS_CS' AND m.column_name = 'SHAPE'
  4   AND c.name = 'cola_c';

NAME
-----
SDO_CS.TRANSFORM(C.SHAPE,M.DIMINFO,8199) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z)
-----
cola_c
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(3.00074114, 3.00291482, 6.00067068, 3.00291287, 6.0006723, 5.00307625, 4.0007
1961, 5.00307838, 3.00074114, 3.00291482))

SQL>
SQL> -- Same as preceding, but using to_sname parameter.
SQL> SELECT c.name, SDO_CS.TRANSFORM(c.shape, m.diminfo, 'Longitude / Latitude
(Arc 1950)')
  2   FROM cola_markets_cs c, user_sdo_geom_metadata m
  3   WHERE m.table_name = 'COLA_MARKETS_CS' AND m.column_name = 'SHAPE'
  4   AND c.name = 'cola_c';

NAME
-----
```

```
SDO_CS.TRANSFORM(C.SHAPE,M.DIMINFO,'LONGITUDE/LATITUDE(ARC1950)')(SDO_GTYPE, SDO
-----
cola_c
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(3.00074114, 3.00291482, 6.00067068, 3.00291287, 6.0006723, 5.00307625, 4.0007
1961, 5.00307838, 3.00074114, 3.00291482))
```

```
SQL>
SQL> -- Transform the entire SHAPE layer and put results in the table
SQL> -- named cola_markets_cs_8199, which the procedure will create.
SQL> CALL SDO_CS.TRANSFORM_LAYER('COLA_MARKETS_CS','SHAPE','COLA_MARKETS_CS_
8199',8199);
```

Call completed.

```
SQL>
SQL> -- Select all from the old (existing) table.
SQL> SELECT * from cola_markets_cs;
```

```

MKT_ID NAME
-----
SHAPE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
1 cola_a
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(1, 1, 5, 1, 5, 7, 1, 7, 1, 1))

2 cola_b
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(5, 1, 8, 1, 8, 6, 5, 7, 5, 1))
```

```

3 cola_c

MKT_ID NAME
-----
SHAPE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(3, 3, 6, 3, 6, 5, 4, 5, 3, 3))

4 cola_d
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(10, 9, 11, 9, 11, 10, 10, 10, 10, 9))
```

```
SQL>
SQL> -- Select all from the new (layer transformed) table.
SQL> SELECT * from cola_markets_cs_8199;
```

```

SDO_ROWID
-----
GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
AAABZzAABAAAOa6AAA
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(1.00078604, 1.00274579, 5.00069354, 1.00274488, 5.0006986, 7.00323528, 1.0007
9179, 7.00324162, 1.00078604, 1.00274579))

AAABZzAABAAAOa6AAB
```

```
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(5.00069354, 1.00274488, 8.00062191, 1.00274427, 8.00062522, 6.00315345, 5.0006986, 7.00323528, 5.00069354, 1.00274488))
```

```
SDO_ROWID
```

```
-----  
GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)  
-----
```

```
AAABZzAABAAA0a6AAC
```

```
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(3.00074114, 3.00291482, 6.00067068, 3.00291287, 6.0006723, 5.00307625, 4.00071961, 5.00307838, 3.00074114, 3.00291482))
```

```
AAABZzAABAAA0a6AAD
```

```
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(10.0005802, 9.00337775, 11.0005553, 9.00337621, 11.0005569, 10.0034478, 10.00
```

```
SDO_ROWID
```

```
-----  
GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)  
-----
```

```
05819, 10.0034495, 10.0005802, 9.00337775))
```

```
SQL>
```

```
SQL> -- Show metadata for the new (layer transformed) table.
```

```
SQL> DESCRIBE cola_markets_cs_8199;
```

Name	Null?	Type
SDO_ROWID		ROWID
GEOMETRY		SDO_GEOMETRY

```
SQL>
```

```
SQL> -- Use a geodetic MBR with SDO_FILTER
```

```
SQL> SELECT c.name FROM cola_markets_cs c WHERE
```

```
2     SDO_FILTER(c.shape,  
3         SDO_GEOMETRY(  
4             2003,  
5             8307,    -- SRID for WGS 84 longitude/latitude  
6             NULL,  
7             SDO_ELEM_INFO_ARRAY(1,1003,3),  
8             SDO_ORDINATE_ARRAY(6,5, 10,10))  
9         ) = 'TRUE';
```

```
NAME
```

```
-----  
cola_c  
cola_b  
cola_d
```

Linear Referencing System

Linear referencing is a natural and convenient means to associate attributes or events to locations or portions of a linear feature. It has been widely used in transportation applications (such as for highways, railroads, and transit routes) and utilities applications (such as for gas and oil pipelines). The major advantage of linear referencing is its capability of locating attributes and events along a linear feature with only one parameter (usually known as *measure*) instead of two (such as *longitude/latitude* or *x/y* in Cartesian space). Sections of a linear feature can be referenced and created dynamically by indicating the start and end locations along the feature without explicitly storing them.

The linear referencing system (LRS) application programming interface (API) in Oracle Spatial provides server-side LRS capabilities at the cartographic level. The linear measure information is directly integrated into the Oracle Spatial geometry structure. The Oracle Spatial LRS API provides support for dynamic segmentation, and it serves as a groundwork for third-party or middle-tier application development for virtually any linear referencing methods and models in any coordinate system.

For an example of LRS, see [Section 7.7](#). However, you may want to read the rest of this chapter first, to understand the concepts that the example illustrates.

For reference information about LRS functions and procedures, see [Chapter 16](#).

If you have LRS data from a previous release of Spatial, see [Section A.1](#) for information about upgrading LRS data.

This chapter contains the following major sections:

- [Section 7.1, "Terms and Concepts"](#)
- [Section 7.2, "LRS Data Model"](#)
- [Section 7.3, "Indexing of LRS Data"](#)
- [Section 7.4, "3D Formats of LRS Functions"](#)
- [Section 7.5, "LRS Operations"](#)
- [Section 7.6, "Tolerance Values with LRS Functions"](#)
- [Section 7.7, "Example of LRS Functions"](#)

7.1 Terms and Concepts

This section explains important terms and concepts related to linear referencing support in Oracle Spatial.

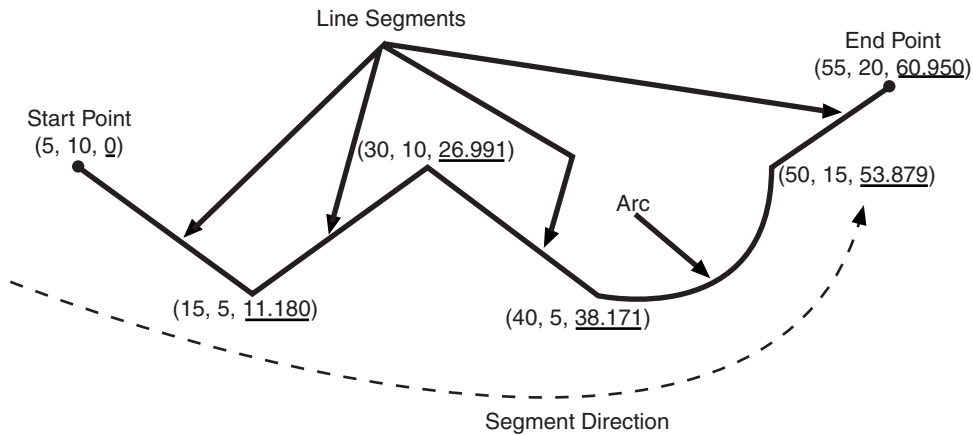
7.1.1 Geometric Segments (LRS Segments)

Geometric segments are basic LRS elements in Oracle Spatial. A geometric segment can be any of the following:

- Line string: an ordered, nonbranching, and continuous geometry (for example, a simple road)
- Multiline string: nonconnected line strings (for example, a highway with a gap caused by a lake or a bypass road)
- Polygon (for example, a racetrack or a scenic tour route that starts and ends at the same point)

A geometric segment must contain at least start and end measures for its start and end points. Measures of points of interest (such as highway exits) on the geometric segments can also be assigned. These measures are either assigned by users or derived from existing geometric segments. [Figure 7-1](#) shows a geometric segment with four line segments and one arc. Points on the geometric segment are represented by triplets (x, y, m) , where x and y describe the location and m denotes the measure (with each measure value underlined in [Figure 7-1](#)).

Figure 7-1 Geometric Segment



7.1.2 Shape Points

Shape points are points that are specified when an LRS segment is constructed, and that are assigned measure information. In Oracle Spatial, a line segment is represented by its start and end points, and an arc is represented by three points: start, middle, and end points of the arc. You must specify these points as shape points, but you can also specify other points as shape points if you need measure information stored for these points (for example, an exit in the middle of a straight part of the highway).

Thus, shape points can serve one or both of the following purposes: to indicate the direction of the segment (for example, a turn or curve), and to identify a point of interest for which measure information is to be stored.

Shape points might not directly relate to mileposts or reference posts in LRS; they are used as internal reference points. The measure information of shape points is automatically populated when you define the LRS segment using the [SDO_LRS.DEFINE_GEOM_SEGMENT](#) procedure, which is described in [Chapter 16](#).

7.1.3 Direction of a Geometric Segment

The **direction** of a geometric segment is indicated from the start point of the geometric segment to the end point. The direction is determined by the order of the vertices (from start point to end point) in the geometry definition. Measures of points on a geometric segment always either increase or decrease along the direction of the geometric segment.

7.1.4 Measure (Linear Measure)

The **measure** of a point along a geometric segment is the linear distance (in the measure dimension) to the point measured from the start point (for increasing values) or end point (for decreasing values) of the geometric segment. The measure information does not necessarily have to be of the same scale as the distance. However, the linear mapping relationship between measure and distance is always preserved.

Some LRS functions use *offset* instead of measure to represent measured distance along linear features. Although some other linear referencing systems might use offset to mean what the Oracle Spatial LRS refers to as measure, offset has a different meaning in Oracle Spatial from measure, as explained in [Section 7.1.5](#).

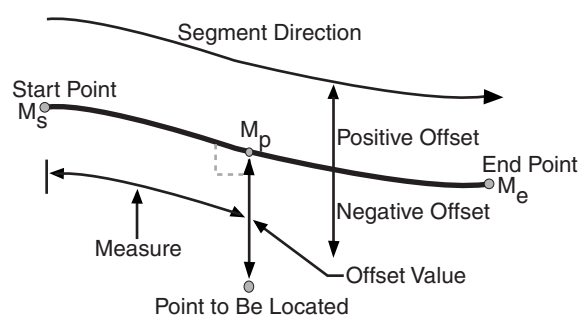
7.1.5 Offset

The **offset** of a point along a geometric segment is the perpendicular distance between the point and the geometric segment. Offsets are positive if the points are on the left side along the segment direction and are negative if they are on the right side. Points are on a geometric segment if their offsets to the segment are zero.

The unit of measurement for an offset is the same as for the coordinate system associated with the geometric segment. For geodetic data, the default unit of measurement is meters.

[Figure 7-2](#) shows how a point can be located along a geometric segment with measure and offset information. By assigning an offset together with a measure, it is possible to locate not only points that are on the geometric segment, but also points that are perpendicular to the geometric segment.

Figure 7-2 Describing a Point Along a Segment with a Measure and an Offset



7.1.6 Measure Populating

Any unassigned measures of a geometric segment are automatically populated based upon their distance distribution. This is done before any LRS operations for geometric segments with unknown measures (NULL in Oracle Spatial). The resulting geometric segments from any LRS operations return the measure information associated with

geometric segments. The measure of a point on the geometric segment can be obtained based upon a linear mapping relationship between its previous and next known measures or locations. See the algorithm representation in [Figure 7-3](#) and the example in [Figure 7-4](#).

Figure 7-3 Measures, Distances, and Their Mapping Relationship

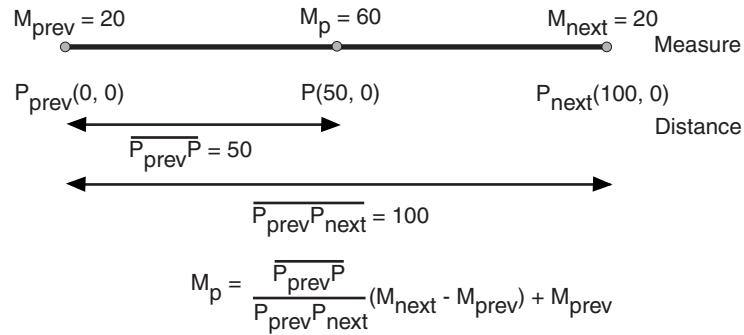
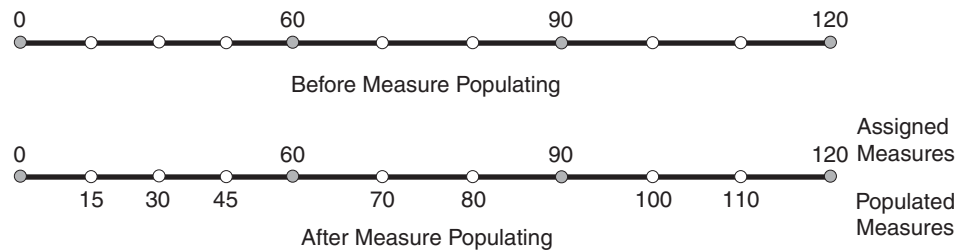


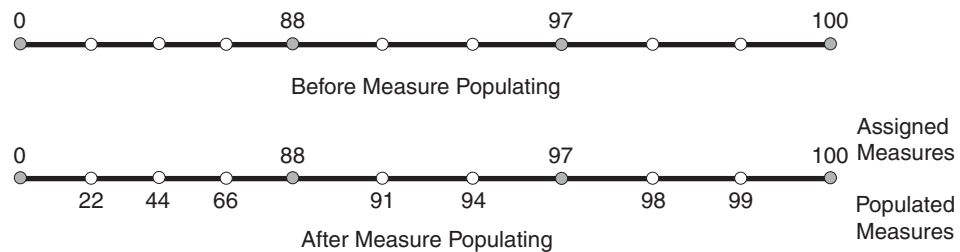
Figure 7-4 Measure Populating of a Geometric Segment



Measures are evenly spaced between assigned measures. However, the assigned measures for points of interest on a geometric segment do not need to be evenly spaced. This could eliminate the problem of error accumulation and account for inaccuracy of data source.

Moreover, the assigned measures do not even need to reflect actual distances (for example, they can reflect estimated driving time); they can be any valid values within the measure range. [Figure 7-5](#) shows the measure population that results when assigned measure values are not proportional and reflect widely varying gaps.

Figure 7-5 Measure Populating with Disproportional Assigned Measures



In all cases, measure populating is done in an incremental fashion along the segment direction. This improves the performance of current and subsequent LRS operations.

7.1.7 Measure Range of a Geometric Segment

The start and end measures of a geometric segment define the linear **measure range** of the geometric segment. Any valid LRS measures of a geometric segment must fall within its linear measure range.

7.1.8 Projection

The **projection** of a point along a geometric segment is the point on the geometric segment with the minimum distance to the specified point. The measure information of the resulting point is also returned in the point geometry.

7.1.9 LRS Point

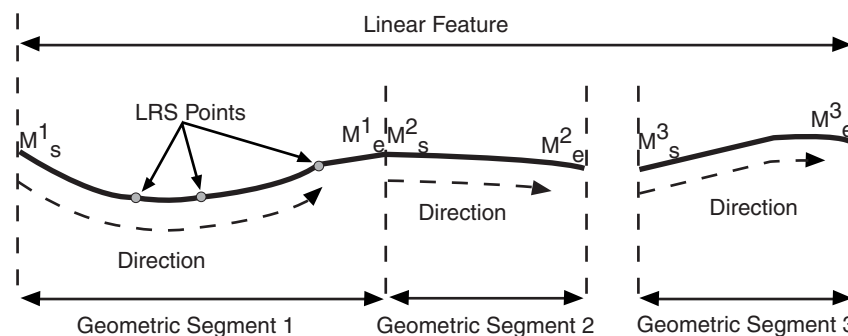
LRS points are points with linear measure information along a geometric segment. A valid LRS point is a point geometry with measure information.

All LRS point data must be stored in the SDO_ELEM_INFO_ARRAY and SDO_ORDINATE_ARRAY, and cannot be stored in the SDO_POINT field in the SDO_GEOMETRY definition of the point.

7.1.10 Linear Features

Linear features are any spatial objects that can be treated as a logical set of linear segments. Examples of linear features are highways in transportation applications and pipelines in utility industry applications. The relationship of linear features, geometric segments, and LRS points is shown in Figure 7–6, where a single linear feature consists of three geometric segments, and three LRS points are shown on the first segment.

Figure 7–6 Linear Feature, Geometric Segments, and LRS Points



7.1.11 Measures with Multiline Strings and Polygons with Holes

With a multiline string or polygon with hole LRS geometry, the [SDO_LRS.DEFINE_GEOM_SEGMENT](#) procedure and [SDO_LRS.CONVERT_TO_LRS_GEOM](#) function by default assign the same measure value to the end point of one segment and the start point (separated by a gap) of the next segment, although you can later assign different measure values to points. Thus, by default there will duplicate measure values in different segments for such geometries. In such cases, LRS subprograms use the first point with a specified measure, except when doing so would result in an invalid geometry.

For example, assume that in a multiline string LRS geometry, the first segment is from measures 0 through 100 and the second segment is from measures 100 through 150. If you use the [SDO_LRS.LOCATE_PT](#) function to find the point at measure 100, the returned point will be at measure 100 in the first segment. If you use the [SDO_LRS.CLIP_GEOM_SEGMENT](#), [SDO_LRS.DYNAMIC_SEGMENT](#), or [SDO_LRS.OFFSET_GEOM_SEGMENT](#) function to return the geometry object between measures 75 and 125, the result is a multiline string geometry consisting of two segments. If you use the same function to return the geometry object between measures 100 and 125, the point at measure 100 in the first segment is ignored, and the result is a line string along the second segment from measures 100 through 125.

7.2 LRS Data Model

The Oracle Spatial LRS data model incorporates measure information into its geometry representation at the point level. The measure information is directly integrated into the Oracle Spatial model. To accomplish this, an additional *measure* dimension must be added to the Oracle Spatial metadata.

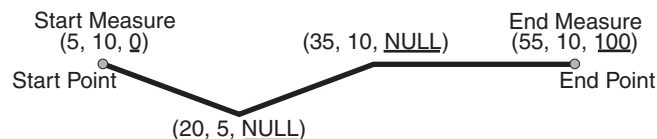
Oracle Spatial LRS support affects the Spatial metadata and data (the geometries). [Example 7-1](#) shows how a measure dimension can be added to two-dimensional geometries in the Spatial metadata. The measure dimension must be the last element of the SDO_DIM_ARRAY in a spatial object definition (shown in bold in [Example 7-1](#)).

Example 7-1 Including LRS Measure Dimension in Spatial Metadata

```
INSERT INTO user_sdo_geom_metadata
  (TABLE_NAME,
   COLUMN_NAME,
   DIMINFO,
   SRID)
VALUES (
  'LRS_ROUTES',
  'GEOMETRY',
  SDO_DIM_ARRAY (
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005),
    SDO_DIM_ELEMENT('M', 0, 100, 0.005)),
  NULL);
```

After adding the new measure dimension, geometries with measure information such as geometric segments and LRS points can be represented. An example of creating a geometric segment with three line segments is shown in [Figure 7-7](#).

Figure 7-7 Creating a Geometric Segment



In [Figure 7-7](#), the geometric segment has the following definition (with measure values underlined):

```
SDO_GEOMETRY(3302, NULL, NULL,
  SDO_ELEM_INFO_ARRAY(1,2,1),
  SDO_ORDINATE_ARRAY(5,10,0, 20,5,NULL, 35,10,NULL, 55,10,100))
```

Whenever a geometric segment is defined, its start and end measures must be defined or derived from some existing geometric segment. The unsigned measures of all shape points on a geometric segment will be automatically populated.

The LRS API works with geometries in formats of Oracle Spatial before release 8.1.6, but the resulting geometries will be converted to the Oracle Spatial release 8.1.6 or later format, specifically with 4-digit SDO_GTYPE and SDO_ETYPE values. For example, in Oracle Spatial release 8.1.6 and later, the geometry type (SDO_GTYPE) of a spatial object includes the number of dimensions of the object as the first digit of the SDO_GTYPE value. Thus, the SDO_GTYPE value of a point is 1 in the format before release 8.1.6 but 2001 in the release 8.1.6 format (the number of dimensions of the point is 2). However, an LRS point (which includes measure information) has three dimensions, and thus the SDO_GTYPE of any point geometry used with an LRS function must be 3301.

7.3 Indexing of LRS Data

If LRS data has four dimensions (three plus the M dimension) and if you need to index all three non-measure dimensions, you must use a spatial R-tree index to index the data, and you must specify `PARAMETERS('sdo_indx_dims=3')` in the `CREATE INDEX` statement to ensure that the first three dimensions are indexed. Note, however, that if you specify an `sdo_indx_dims` value of 3 or higher, the only Spatial operator that can be used on the indexed geometries is `SDO_FILTER`; the other operators described in [Chapter 11](#) cannot be used. (The default value for the `sdo_indx_dims` keyword is 2, which would cause only the first two dimensions to be indexed.) For example, if the dimensions are X, Y, Z, and M, specify `sdo_indx_dims=3` to index the X, Y, and Z dimensions, but not the measure (M) dimension. Do not include the measure dimension in a spatial index, because this causes additional processing overhead and produces no benefit.

Information about the `CREATE INDEX` statement and its parameters and keywords is in [Chapter 10](#).

7.4 3D Formats of LRS Functions

Most LRS functions have formats that end in `_3D`: for example, `DEFINE_GEOM_SEGMENT_3D`, `CLIP_GEOM_SEGMENT_3D`, `FIND_MEASURE_3D`, and `LOCATE_PT_3D`. If a function has a `3D` format, it is identified in the Usage Notes for the function in [Chapter 16](#).

The `3D` formats are supported only for line string and multiline string geometries. The `3D` formats should be used only when the geometry object has four dimensions and the fourth dimension is the measure (for example, X, Y, Z, and M), and only when you want the function to consider the first three dimensions (for example, X, Y, and Z). If the standard format of a function (that is, without the `_3D`) is used on a geometry with four dimensions, the function considers only the first two dimensions (for example, X and Y).

For example, the following format considers the X, Y, and Z dimensions of the specified GEOM object in performing the clip operation:

```
SELECT SDO_LRS.CLIP_GEOM_SEGMENT_3D(a.geom, m.dinfo, 5, 10)
FROM routes r, user_sdo_geom_metadata m
WHERE m.table_name = 'ROUTES' AND m.column_name = 'GEOM'
AND r.route_id = 1;
```

However, the following format considers only the X and Y dimensions, and ignores the Z dimension, of the specified GEOM object in performing the clip operation:

```
SELECT SDO_LRS.CLIP_GEOM_SEGMENT(a.geom, m.diminfo, 5, 10)
FROM routes r, user_sdo_geom_metadata m
WHERE m.table_name = 'ROUTES' AND m.column_name = 'GEOM'
AND r.route_id = 1;
```

The parameters for the standard and 3D formats of any function are the same, and the Usage Notes apply to both formats.

The 3D formats are not supported with the following:

- Geodetic data
- Polygons, arcs, or circles

7.5 LRS Operations

This section describes several linear referencing operations supported by the Oracle Spatial LRS API.

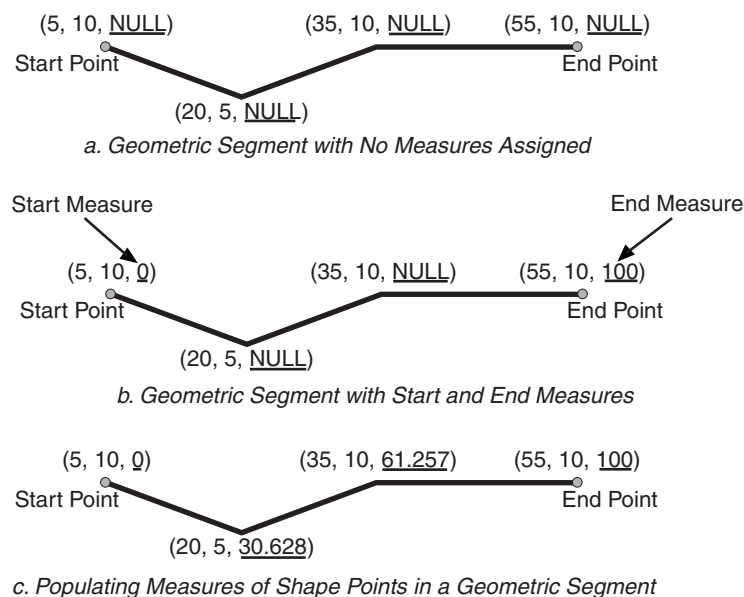
7.5.1 Defining a Geometric Segment

There are two ways to create a geometric segment with measure information:

- Construct a geometric segment and assign measures explicitly.
- Define a geometric segment with specified start and end, and any other measures, in an ascending or descending order. Measures of shape points with unknown (unassigned) measures (null values) in the geometric segment will be automatically populated according to their locations and distance distribution.

Figure 7–8 shows different ways of defining a geometric segment:

Figure 7–8 Defining a Geometric Segment



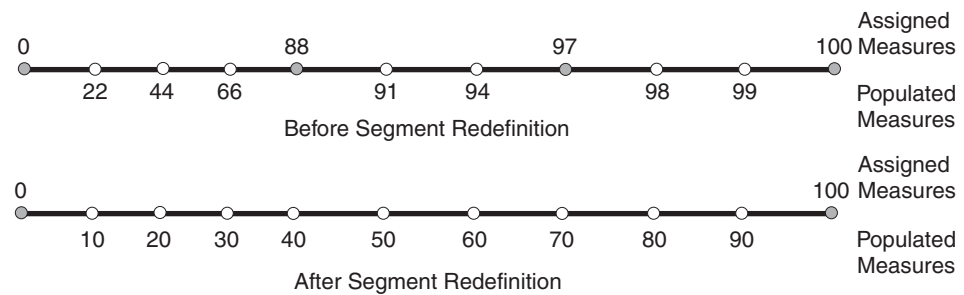
An LRS segment must be defined (or must already exist) before any LRS operations can proceed. That is, the start, end, and any other assigned measures must be present to derive the location from a specified measure. The measure information of intermediate shape points will automatically be populated if measure values are not assigned.

7.5.2 Redefining a Geometric Segment

You can redefine a geometric segment to replace the existing measures of all shape points between the start and end point with automatically calculated measures. Redefining a segment can be useful if errors have been made in one or more explicit measure assignments, and you want to start over with proportionally assigned measures.

Figure 7-9 shows the redefinition of a segment where the existing (before) assigned measure values are not proportional and reflect widely varying gaps.

Figure 7-9 Redefining a Geometric Segment

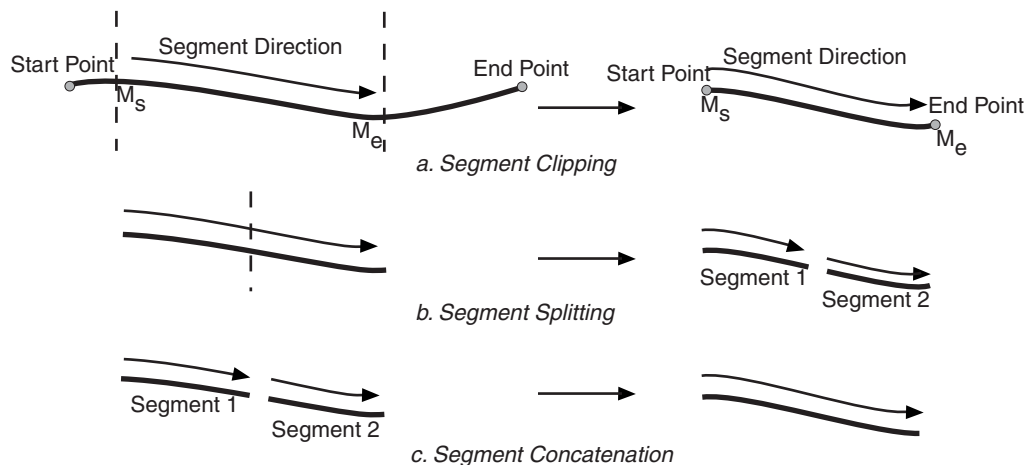


After the segment redefinition in Figure 7-9, the populated measures reflect proportional distances along the segment.

7.5.3 Clipping a Geometric Segment

You can clip a geometric segment to create a new geometric segment out of an existing geometric segment, as shown in Figure 7-10, part a.

Figure 7-10 Clipping, Splitting, and Concatenating Geometric Segments



In [Figure 7–10](#), part a, a segment is created from part of a larger segment. The new segment has its own start and end points, and the direction is the same as in the original larger segment.

7.5.4 Splitting a Geometric Segment

You can create two new geometric segments by splitting a geometric segment, as shown in [Figure 7–10](#), part b. The direction of each new segment is the same as in the original segment.

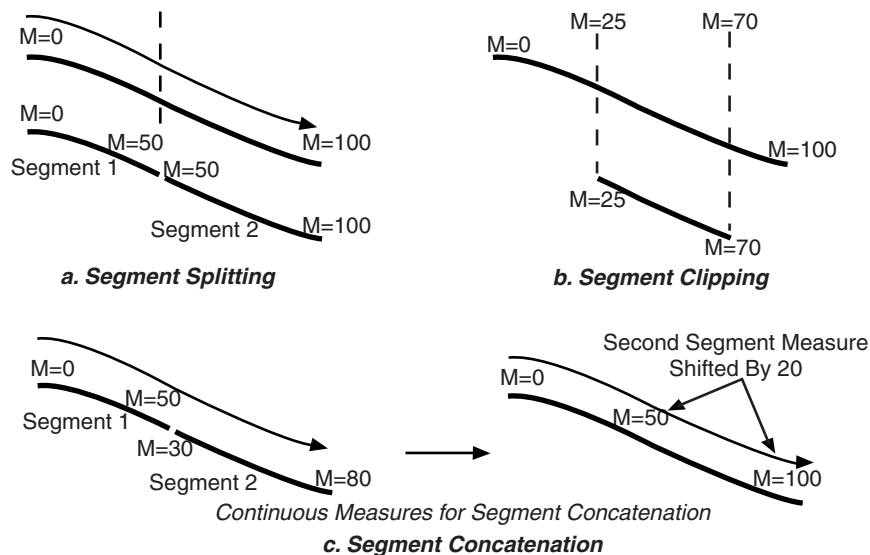
Note: In [Figure 7–10](#) and several figures that follow, small gaps between segments are used in illustrations of segment splitting and concatenation. Each gap simply reinforces the fact that two different segments are involved. However, the two segments (such as segment 1 and segment 2 in [Figure 7–10](#), parts b and c) are actually connected. The tolerance (see [Section 1.5.5](#)) is considered in determining whether or not segments are connected.

7.5.5 Concatenating Geometric Segments

You can create a new geometric segment by concatenating two geometric segments, as shown in [Figure 7–10](#), part c. The geometric segments do not need to be spatially connected, although they are connected in the illustration in [Figure 7–10](#), part c. (If the segments are not spatially connected, the concatenated result is a multiline string.) The measures of the second geometric segment are shifted so that the end measure of the first segment is the same as the start measure of the second segment. The direction of the segment resulting from the concatenation is the same as in the two original segments.

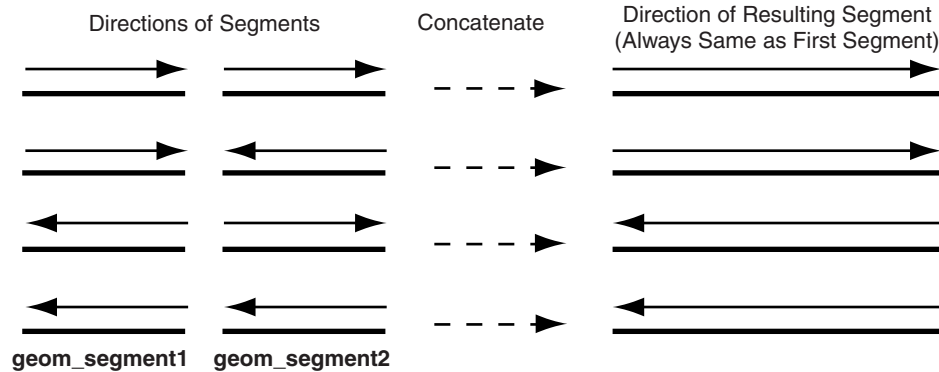
Measure assignments for the clipping, splitting, and concatenating operations in [Figure 7–10](#) are shown in [Figure 7–11](#). Measure information and segment direction are preserved in a consistent manner. The assignment is done automatically when the operations have completed.

Figure 7–11 Measure Assignment in Geometric Segment Operations



The direction of the geometric segment resulting from concatenation is always the direction of the first segment (`geom_segment1` in the call to the `SDO_LRS.CONCATENATE_GEOM_SEGMENTS` function), as shown in Figure 7–12.

Figure 7–12 Segment Direction with Concatenation

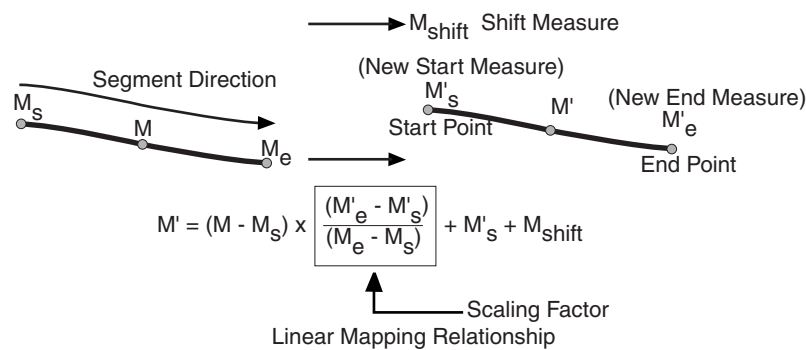


In addition to explicitly concatenating two connected segments using the `SDO_LRS.CONCATENATE_GEOM_SEGMENTS` function, you can perform aggregate concatenation: that is, you can concatenate all connected geometric segments in a column (layer) using the `SDO_AGGR_LRS_CONCAT` spatial aggregate function. (See the description and example of the `SDO_AGGR_LRS_CONCAT` spatial aggregate function in Chapter 12.)

7.5.6 Scaling a Geometric Segment

You can create a new geometric segment by performing a linear scaling operation on a geometric segment. Figure 7–13 shows the mapping relationship for geometric segment scaling.

Figure 7–13 Scaling a Geometric Segment



In general, scaling a geometric segment only involves rearranging measures of the newly created geometric segment. However, if the scaling factor is negative, the order of the shape points needs to be reversed so that measures will increase along the geometric segment's direction (which is defined by the order of the shape points).

A scale operation can perform any combination of the following operations:

- Translating (shifting) measure information. (For example, add the same value to M_s and M_e to get M'_s and M'_e .)

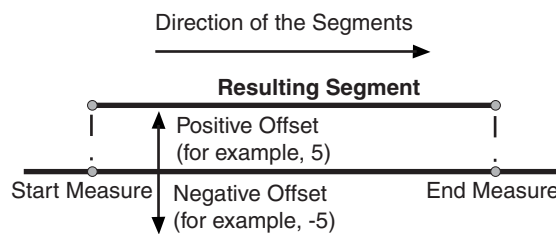
- Reversing measure information. (Let $M's = M_e$, $M'e = M_s$, and $Mshift = 0$.)
- Performing simple scaling of measure information. (Let $Mshift = 0$.)

For examples of these operations, see the Usage Notes and Examples for the [SDO_LRS.TRANSLATE_MEASURE](#), [SDO_LRS.REVERSE_GEOMETRY](#), and [SDO_LRS.REDEFINE_GEOM_SEGMENT](#) subprograms in [Chapter 16](#).

7.5.7 Offsetting a Geometric Segment

You can create a new geometric segment by performing an offsetting operation on a geometric segment. [Figure 7-14](#) shows the mapping relationship for geometric segment offsetting.

Figure 7-14 Offsetting a Geometric Segment



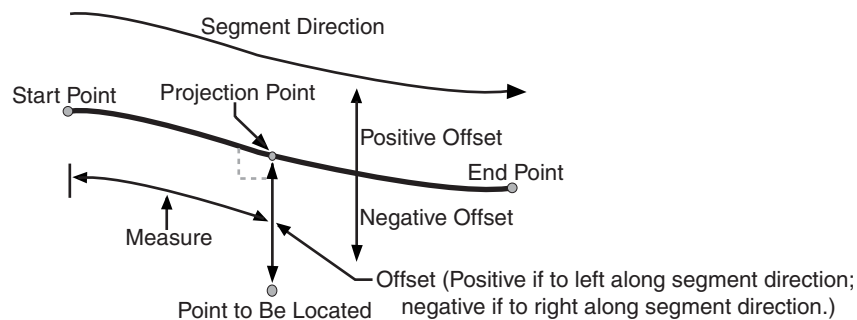
In the offsetting operation shown in [Figure 7-14](#), the resulting geometric segment is offset by 5 units from the specified start and end measures of the original segment.

For more information, see the Usage Notes and Examples for the [SDO_LRS.OFFSET_GEOM_SEGMENT](#) function in [Chapter 16](#).

7.5.8 Locating a Point on a Geometric Segment

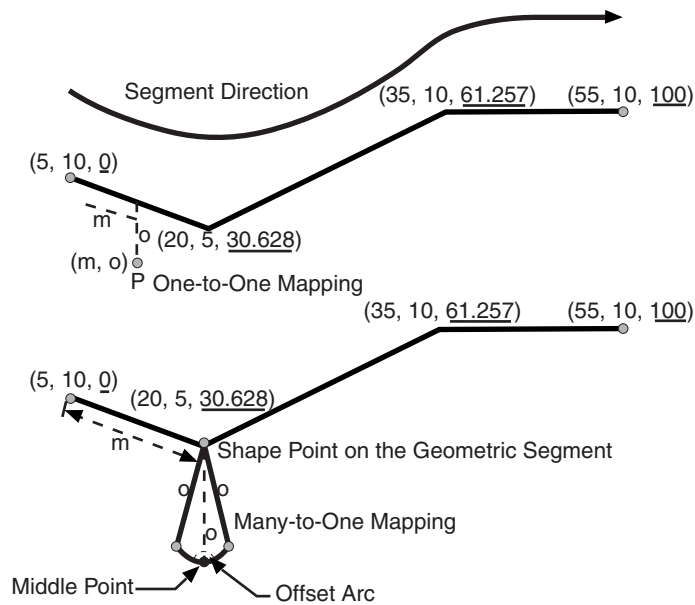
You can find the position of a point described by a measure and an offset on a geometric segment (see [Figure 7-15](#)).

Figure 7-15 Locating a Point Along a Segment with a Measure and an Offset



There is always a unique location with a specific measure on a geometric segment. Ambiguity arises when offsets are given and the points described by the measures fall on shape points of the geometric segment (see [Figure 7-16](#)).

Figure 7–16 Ambiguity in Location Referencing with Offsets



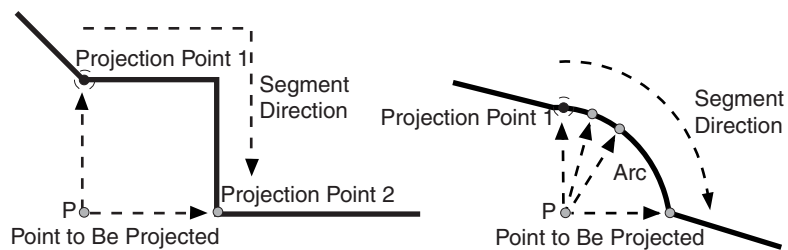
As shown in [Figure 7–16](#), an offset arc of a shape point on a geometric segment is an arc on which all points have the same minimum distance to the shape point. As a result, all points on the offset arc are represented by the same (measure, offset) pair. To resolve this one-to-many mapping problem, the middle point on the offset arc is returned.

7.5.9 Projecting a Point onto a Geometric Segment

You can find the projection point of a point with respect to a geometric segment. The point to be projected can be on or off the segment. If the point is on the segment, the point and its projection point are the same.

Projection is a reverse operation of the point-locating operation shown in [Figure 7–15](#). Similar to a point-locating operation, all points on the offset arc of a shape point will have the same projection point (that is, the shape point itself), measure, and offset (see [Figure 7–16](#)). If there are multiple projection points for a point, the first one from the start point is returned (Projection Point 1 in both illustrations in [Figure 7–17](#)).

Figure 7–17 Multiple Projection Points



7.5.10 Converting LRS Geometries

You can convert geometries from standard line string format to LRS format, and the reverse. The main use of conversion functions will probably occur if you have a large amount of existing line string data, in which case conversion is a convenient alternative to creating all of the LRS segments manually. However, if you need to convert LRS segments to standard line strings for certain applications, that capability is provided also.

Functions are provided to convert:

- Individual line strings or points

For conversion from standard format to LRS format, a measure dimension (named *M* by default) is added, and measure information is provided for each point. For conversion from LRS format to standard format, the measure dimension and information are removed. In both cases, the dimensional information (DIMINFO) metadata in the USER_SDO_GEOM_METADATA view is not affected.

- Layers (all geometries in a column)

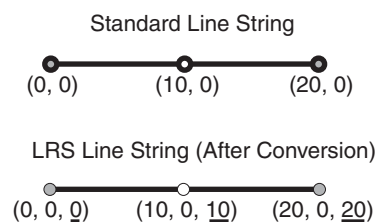
For conversion from standard format to LRS format, a measure dimension (named *M* by default) is added, but no measure information is provided for each point. For conversion from LRS format to standard format, the measure dimension and information are removed. In both cases, the dimensional information (DIMINFO) metadata in the USER_SDO_GEOM_METADATA view is modified as needed.

- Dimensional information (DIMINFO)

The dimensional information (DIMINFO) metadata in the USER_SDO_GEOM_METADATA view is modified as needed. For example, converting a standard dimensional array with X and Y dimensions (SDO_DIM_ELEMENT) to an LRS dimensional array causes an M dimension (SDO_DIM_ELEMENT) to be added.

Figure 7–18 shows the addition of measure information when a standard line string is converted to an LRS line string (using the `SDO_LRS.CONVERT_TO_LRS_GEOM` function). The measure dimension values are underlined in Figure 7–18.

Figure 7–18 Conversion from Standard to LRS Line String



For conversions of point geometries, the SDO_POINT attribute (described in Section 2.2.3) in the returned geometry is affected as follows:

- If a standard point is converted to an LRS point, the SDO_POINT attribute information in the input geometry is used to set the SDO_ELEM_INFO and SDO_ORDINATES attributes (described in Section 2.2.4 and Section 2.2.5) in the resulting geometry, and the SDO_POINT attribute in the resulting geometry is set to null.
- If an LRS point is converted to a standard point, the information in the SDO_ELEM_INFO and SDO_ORDINATES attributes (described in Section 2.2.4 and Section 2.2.5) in the input geometry is used to set the SDO_POINT attribute

information in the resulting geometry, and the SDO_ELEM_INFO and SDO_ORDINATES attributes in the resulting geometry are set to null.

The conversion functions are listed in [Table 16–3](#) in [Chapter 16](#). See also the reference information in [Chapter 16](#) about each conversion function.

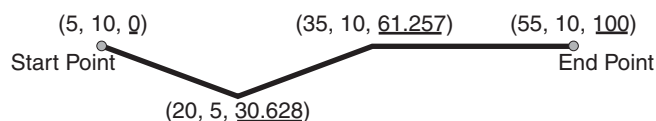
7.6 Tolerance Values with LRS Functions

Many LRS functions require that you specify a tolerance value or one or more dimensional arrays. Thus, you can control whether to specify a single tolerance value for all non-measure dimensions or to use the tolerance associated with each non-measure dimension in the dimensional array or arrays. The tolerance is applied only to the geometry portion of the data, not to the measure dimension. The tolerance value for geodetic data is in meters, and for non-geodetic data it is in the unit of measurement associated with the data. (For a detailed discussion of tolerance, see [Section 1.5.5](#).)

Be sure that the tolerance value used is appropriate to the data and your purpose. If the results of LRS functions seem imprecise or incorrect, you may need to specify a smaller tolerance value.

For clip operations (see [Section 7.5.3](#)) and offset operations (see [Section 7.5.7](#)), if the returned segment has any shape points within the tolerance value of the input geometric segment from what would otherwise be the start point or end point of the returned segment, the shape point is used as the start point or end point of the returned segment. This is done to ensure that the resulting geometry does not contain any redundant vertices, which would cause the geometry to be invalid. For example, assume that the tolerance associated with the geometric segment (non-geodetic data) in [Figure 7–19](#) is 0.5.

Figure 7–19 Segment for Clip Operation Affected by Tolerance



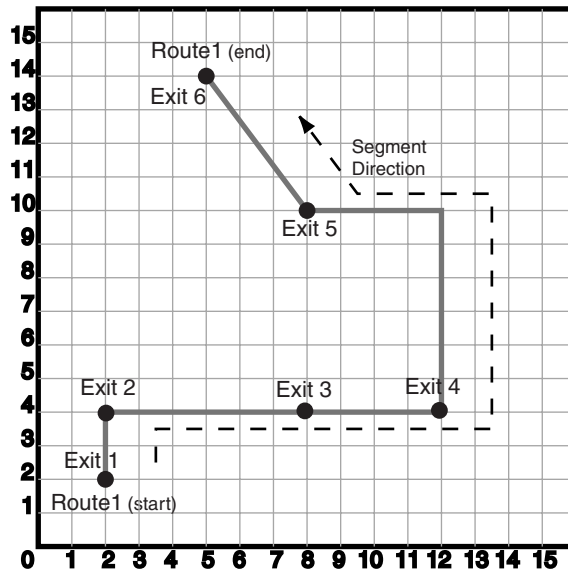
If you request a clip operation to return the segment between measure values 0 (the start point) and 61.5 in [Figure 7–19](#), and if the distance between the points associated with measure values 61.5 and 61.257 is less than the 0.5 tolerance value, the end point of the returned segment is (35, 10, 61.257).

7.7 Example of LRS Functions

This section presents a simplified example that uses LRS functions. It refers to concepts that are explained in this chapter and uses functions documented in [Chapter 16](#).

This example uses the road that is illustrated in [Figure 7–20](#).

Figure 7–20 Simplified LRS Example: Highway



In [Figure 7–20](#), the highway (Route 1) starts at point 2,2 and ends at point 5,14, follows the path shown, and has six entrance-exit points (Exit 1 through Exit 6). For simplicity, each unit on the graph represents one unit of measure, and thus the measure from start to end is 27 (the segment from Exit 5 to Exit 6 being the hypotenuse of a 3-4-5 right triangle).

Each row in [Table 7–1](#) lists an actual highway-related feature and the LRS feature that corresponds to it or that can be used to represent it.

Table 7–1 Highway Features and LRS Counterparts

Highway Feature	LRS Feature
Named route, road, or street	LRS segment, or linear feature (logical set of segments)
Mile or kilometer marker	Measure
Accident reporting and location tracking	SDO_LRS.LOCATE_PT function
Construction zone (portion of a road)	SDO_LRS.CLIP_GEOM_SEGMENT function
Road extension (adding at the beginning or end) or combination (designating or renaming two roads that meet as one road)	SDO_LRS.CONCATENATE_GEOM_SEGMENTS function
Road reconstruction or splitting (resulting in two named roads from one named road)	SDO_LRS.SPLIT_GEOM_SEGMENT procedure
Finding the closest point on the road to a point off the road (such as a building)	SDO_LRS.PROJECT_PT function
Guard rail or fence alongside a road	SDO_LRS.OFFSET_GEOM_SEGMENT function

[Example 7–2](#) does the following:

- Creates a table to hold the segment depicted in [Figure 7–20](#)
- Inserts the definition of the highway depicted in [Figure 7–20](#) into the table

- Inserts the necessary metadata into the USER_SDO_GEOM_METADATA view
- Uses PL/SQL and SQL statements to define the segment and perform operations on it

Example 7-3 includes the output of the SELECT statements in Example 7-2.

Example 7-2 Simplified Example: Highway

```
-- Create a table for routes (highways).
CREATE TABLE lrs_routes (
  route_id NUMBER PRIMARY KEY,
  route_name VARCHAR2(32),
  route_geometry SDO_GEOMETRY);

-- Populate table with just one route for this example.
INSERT INTO lrs_routes VALUES(
  1,
  'Route1',
  SDO_GEOMETRY(
    3302, -- line string, 3 dimensions: X,Y,M
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1), -- one line string, straight segments
    SDO_ORDINATE_ARRAY(
      2,2,0, -- Start point - Exit1; 0 is measure from start.
      2,4,2, -- Exit2; 2 is measure from start.
      8,4,8, -- Exit3; 8 is measure from start.
      12,4,12, -- Exit4; 12 is measure from start.
      12,10,NULL, -- Not an exit; measure automatically calculated and filled.
      8,10,22, -- Exit5; 22 is measure from start.
      5,14,27) -- End point (Exit6); 27 is measure from start.
    )
  );

-- Update the Spatial metadata.
INSERT INTO user_sdo_geom_metadata
  (TABLE_NAME,
   COLUMN_NAME,
   DIMINFO,
   SRID)
VALUES (
  'lrs_routes',
  'route_geometry',
  SDO_DIM_ARRAY( -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005),
    SDO_DIM_ELEMENT('M', 0, 20, 0.005) -- Measure dimension
  ),
  NULL -- SRID
);

-- Create the spatial index.
CREATE INDEX lrs_routes_idx ON lrs_routes(route_geometry)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;

-- Test the LRS procedures.
DECLARE
geom_segment SDO_GEOMETRY;
line_string SDO_GEOMETRY;
dim_array SDO_DIM_ARRAY;
```

```

result_geom_1 SDO_GEOMETRY;
result_geom_2 SDO_GEOMETRY;
result_geom_3 SDO_GEOMETRY;

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
  WHERE a.route_name = 'Route1';
SELECT m.diminfo into dim_array from
  user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

-- Define the LRS segment for Route1. This will populate any null measures.
-- No need to specify start and end measures, because they are already defined
-- in the geometry.
SDO_LRS.DEFINE_GEOM_SEGMENT (geom_segment, dim_array);

SELECT a.route_geometry INTO line_string FROM lrs_routes a
  WHERE a.route_name = 'Route1';

-- Split Route1 into two segments.
SDO_LRS.SPLIT_GEOM_SEGMENT(line_string,dim_array,5,result_geom_1,result_geom_2);

-- Concatenate the segments that were just split.
result_geom_3 := SDO_LRS.CONCATENATE_GEOM_SEGMENTS(result_geom_1, dim_array,
result_geom_2, dim_array);

-- Update and insert geometries into table, to display later.
UPDATE lrs_routes a SET a.route_geometry = geom_segment
  WHERE a.route_id = 1;

INSERT INTO lrs_routes VALUES(
  11,
  'result_geom_1',
  result_geom_1
);
INSERT INTO lrs_routes VALUES(
  12,
  'result_geom_2',
  result_geom_2
);
INSERT INTO lrs_routes VALUES(
  13,
  'result_geom_3',
  result_geom_3
);

END;
/

-- First, display the data in the LRS table.
SELECT route_id, route_name, route_geometry FROM lrs_routes;

-- Are result_geom_1 and result_geom2 connected?
SELECT  SDO_LRS.CONNECTED_GEOM_SEGMENTS(a.route_geometry,
      b.route_geometry, 0.005)
  FROM lrs_routes a, lrs_routes b
  WHERE a.route_id = 11 AND b.route_id = 12;

-- Is the Route1 segment valid?

```



```

SELECT SDO_LRS.VALID_GEOM_SEGMENT(route_geometry)
  FROM lrs_routes WHERE route_id = 1;

-- Is 50 a valid measure on Route1? (Should return FALSE; highest Route1 measure
is 27.)
SELECT SDO_LRS.VALID_MEASURE(route_geometry, 50)
  FROM lrs_routes WHERE route_id = 1;

-- Is the Route1 segment defined?
SELECT SDO_LRS.IS_GEOM_SEGMENT_DEFINED(route_geometry)
  FROM lrs_routes WHERE route_id = 1;

-- How long is Route1?
SELECT SDO_LRS.GEOM_SEGMENT_LENGTH(route_geometry)
  FROM lrs_routes WHERE route_id = 1;

-- What is the start measure of Route1?
SELECT SDO_LRS.GEOM_SEGMENT_START_MEASURE(route_geometry)
  FROM lrs_routes WHERE route_id = 1;

-- What is the end measure of Route1?
SELECT SDO_LRS.GEOM_SEGMENT_END_MEASURE(route_geometry)
  FROM lrs_routes WHERE route_id = 1;

-- What is the start point of Route1?
SELECT SDO_LRS.GEOM_SEGMENT_START_PT(route_geometry)
  FROM lrs_routes WHERE route_id = 1;

-- What is the end point of Route1?
SELECT SDO_LRS.GEOM_SEGMENT_END_PT(route_geometry)
  FROM lrs_routes WHERE route_id = 1;

-- Translate (shift measure values) (+10).
-- First, display the original segment; then, translate.
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;
SELECT SDO_LRS.TRANSLATE_MEASURE(a.route_geometry, m.diminfo, 10)
  FROM lrs_routes a, user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
  AND a.route_id = 1;

-- Redefine geometric segment to "convert" miles to kilometers
DECLARE
geom_segment SDO_GEOMETRY;
dim_array SDO_DIM_ARRAY;

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
  WHERE a.route_name = 'Route1';
SELECT m.diminfo into dim_array from
  user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

-- "Convert" mile measures to kilometers (27 * 1.609 = 43.443).
SDO_LRS.REDEFINE_GEOM_SEGMENT (geom_segment,
  dim_array,
  0, -- Zero starting measure: LRS segment starts at start of route.
  43.443); -- End of LRS segment. 27 miles = 43.443 kilometers.

-- Update and insert geometries into table, to display later.

```

```

UPDATE lrs_routes a SET a.route_geometry = geom_segment
  WHERE a.route_id = 1;

END;
/
-- Display the redefined segment, with all measures "converted."
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

-- Clip a piece of Route1.
SELECT SDO_LRS.CLIP_GEOM_SEGMENT(route_geometry, 5, 10)
  FROM lrs_routes WHERE route_id = 1;

-- Point (9,3,NULL) is off the road; should return (9,4,9).
SELECT SDO_LRS.PROJECT_PT(route_geometry,
  SDO_GEOMETRY(3301, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1, 1, 1),
    SDO_ORDINATE_ARRAY(9, 3, NULL)) )
  FROM lrs_routes WHERE route_id = 1;

-- Return the measure of the projected point.
SELECT SDO_LRS.GET_MEASURE(
  SDO_LRS.PROJECT_PT(a.route_geometry, m.diminfo,
    SDO_GEOMETRY(3301, NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1, 1, 1),
      SDO_ORDINATE_ARRAY(9, 3, NULL)) ),
  m.diminfo )
  FROM lrs_routes a, user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
    AND a.route_id = 1;

-- Is point (9,3,NULL) a valid LRS point? (Should return TRUE.)
SELECT SDO_LRS.VALID_LRS_PT(
  SDO_GEOMETRY(3301, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1, 1, 1),
    SDO_ORDINATE_ARRAY(9, 3, NULL)),
  m.diminfo)
  FROM lrs_routes a, user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
    AND a.route_id = 1;

-- Locate the point on Route1 at measure 9, offset 0.
SELECT SDO_LRS.LOCATE_PT(route_geometry, 9, 0)
  FROM lrs_routes WHERE route_id = 1;

```

[Example 7-3](#) shows the output of the SELECT statements in [Example 7-2](#).

Example 7-3 Simplified Example: Output of SELECT Statements

```

SQL> -- First, display the data in the LRS table.
SQL> SELECT route_id, route_name, route_geometry FROM lrs_routes;

  ROUTE_ID ROUTE_NAME
-----
ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
          1 Route1
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))

          11 result_geom_1

```

```
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 5, 4, 5))
```

```
12 result_geom_2
```

```
ROUTE_ID ROUTE_NAME
```

```
-----
ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
5, 4, 5, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))
```

```
13 result_geom_3
```

```
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 5, 4, 5, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27)
)
```

```
SQL> -- Are result_geom_1 and result_geom2 connected?
```

```
SQL> SELECT SDO_LRS.CONNECTED_GEOM_SEGMENTS(a.route_geometry,
2 b.route_geometry, 0.005)
3 FROM lrs_routes a, lrs_routes b
4 WHERE a.route_id = 11 AND b.route_id = 12;
```

```
SDO_LRS.CONNECTED_GEOM_SEGMENTS(A.ROUTE_GEOMETRY,B.ROUTE_GEOMETRY,0.005)
```

```
-----
TRUE
```

```
SQL> -- Is the Route1 segment valid?
```

```
SQL> SELECT SDO_LRS.VALID_GEOM_SEGMENT(route_geometry)
2 FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.VALID_GEOM_SEGMENT(ROUTE_GEOMETRY)
```

```
-----
TRUE
```

```
SQL> -- Is 50 a valid measure on Route1? (Should return FALSE; highest Route1
measure is 27.)
```

```
SQL> SELECT SDO_LRS.VALID_MEASURE(route_geometry, 50)
2 FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.VALID_MEASURE(ROUTE_GEOMETRY,50)
```

```
-----
FALSE
```

```
SQL> -- Is the Route1 segment defined?
```

```
SQL> SELECT SDO_LRS.IS_GEOM_SEGMENT_DEFINED(route_geometry)
2 FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.IS_GEOM_SEGMENT_DEFINED(ROUTE_GEOMETRY)
```

```
-----
TRUE
```

```
SQL> -- How long is Route1?
```

```
SQL> SELECT SDO_LRS.GEOM_SEGMENT_LENGTH(route_geometry)
2 FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.GEOM_SEGMENT_LENGTH(ROUTE_GEOMETRY)
```

```
-----
27
```

```

SQL> -- What is the start measure of Route1?
SQL> SELECT SDO_LRS.GEOM_SEGMENT_START_MEASURE(route_geometry)
      2 FROM lrs_routes WHERE route_id = 1;

SDO_LRS.GEOM_SEGMENT_START_MEASURE(ROUTE_GEOMETRY)
-----
0

SQL> -- What is the end measure of Route1?
SQL> SELECT SDO_LRS.GEOM_SEGMENT_END_MEASURE(route_geometry)
      2 FROM lrs_routes WHERE route_id = 1;

SDO_LRS.GEOM_SEGMENT_END_MEASURE(ROUTE_GEOMETRY)
-----
27

SQL> -- What is the start point of Route1?
SQL> SELECT SDO_LRS.GEOM_SEGMENT_START_PT(route_geometry)
      2 FROM lrs_routes WHERE route_id = 1;

SDO_LRS.GEOM_SEGMENT_START_PT(ROUTE_GEOMETRY) (SDO_GTYPE, SDO_SRID, SDO_POINT(X,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
2, 2, 0))

SQL> -- What is the end point of Route1?
SQL> SELECT SDO_LRS.GEOM_SEGMENT_END_PT(route_geometry)
      2 FROM lrs_routes WHERE route_id = 1;

SDO_LRS.GEOM_SEGMENT_END_PT(ROUTE_GEOMETRY) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
5, 14, 27))

SQL> -- Translate (shift measure values) (+10).
SQL> -- First, display the original segment; then, translate.
SQL> SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))

SQL> SELECT SDO_LRS.TRANSLATE_MEASURE(a.route_geometry, m.diminfo, 10)
      2 FROM lrs_routes a, user_sdo_geom_metadata m
      3 WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
      4 AND a.route_id = 1;

SDO_LRS.TRANSLATE_MEASURE(A.ROUTE_GEOMETRY,M.DIMINFO,10) (SDO_GTYPE, SDO_SRID, SD
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 10, 2, 4, 12, 8, 4, 18, 12, 4, 22, 12, 10, 28, 8, 10, 32, 5, 14, 37))

SQL> -- Redefine geometric segment to "convert" miles to kilometers
SQL> DECLARE
      2 geom_segment SDO_GEOMETRY;
      3 dim_array SDO_DIM_ARRAY;
      4
      5 BEGIN

```

```

6
7  SELECT a.route_geometry into geom_segment FROM lrs_routes a
8     WHERE a.route_name = 'Route1';
9  SELECT m.diminfo into dim_array from
10     user_sdo_geom_metadata m
11     WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';
12
13  -- "Convert" mile measures to kilometers (27 * 1.609 = 43.443).
14  SDO_LRS.REDEFINE_GEOM_SEGMENT (geom_segment,
15     dim_array,
16     0, -- Zero starting measure: LRS segment starts at start of route.
17     43.443); -- End of LRS segment. 27 miles = 43.443 kilometers.
18
19  -- Update and insert geometries into table, to display later.
20  UPDATE lrs_routes a SET a.route_geometry = geom_segment
21     WHERE a.route_id = 1;
22
23  END;
24  /

```

PL/SQL procedure successfully completed.

```

SQL> -- Display the redefined segment, with all measures "converted."
SQL> SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

```

```

ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 3.218, 8, 4, 12.872, 12, 4, 19.308, 12, 10, 28.962, 8, 10, 35.398
, 5, 14, 43.443))

```

```

SQL> -- Clip a piece of Route1.

```

```

SQL> SELECT SDO_LRS.CLIP_GEOM_SEGMENT(route_geometry, 5, 10)
2     FROM lrs_routes WHERE route_id = 1;

```

```

SDO_LRS.CLIP_GEOM_SEGMENT(ROUTE_GEOMETRY,5,10)(SDO_GTYPE, SDO_SRID, SDO_POINT(X,
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
5, 4, 5, 8, 4, 8, 10, 4, 10))

```

```

SQL> -- Point (9,3,NULL) is off the road; should return (9,4,9).

```

```

SQL> SELECT SDO_LRS.PROJECT_PT(route_geometry,
2     SDO_GEOMETRY(3301, NULL, NULL,
3     SDO_ELEM_INFO_ARRAY(1, 1, 1),
4     SDO_ORDINATE_ARRAY(9, 3, NULL)) )
5     FROM lrs_routes WHERE route_id = 1;

```

```

SDO_LRS.PROJECT_PT(ROUTE_GEOMETRY,SDO_GEOMETRY(3301,NULL,NULL,SDO_EL
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
9, 4, 9))

```

```

SQL> -- Return the measure of the projected point.

```

```

SQL> SELECT SDO_LRS.GET_MEASURE(
2     SDO_LRS.PROJECT_PT(a.route_geometry, m.diminfo,
3     SDO_GEOMETRY(3301, NULL, NULL,
4     SDO_ELEM_INFO_ARRAY(1, 1, 1),
5     SDO_ORDINATE_ARRAY(9, 3, NULL)) ),
6     m.diminfo )
7     FROM lrs_routes a, user_sdo_geom_metadata m

```

```

8 WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
9 AND a.route_id = 1;

SDO_LRS.GET_MEASURE(SDO_LRS.PROJECT_PT(A.ROUTE_GEOMETRY,M.DIMINFO,SDO_GEOM
-----
9

SQL> -- Is point (9,3,NULL) a valid LRS point? (Should return TRUE.)
SQL> SELECT SDO_LRS.VALID_LRS_PT(
2 SDO_GEOMETRY(3301, NULL, NULL,
3 SDO_ELEM_INFO_ARRAY(1, 1, 1),
4 SDO_ORDINATE_ARRAY(9, 3, NULL)),
5 m.diminfo)
6 FROM lrs_routes a, user_sdo_geom_metadata m
7 WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
8 AND a.route_id = 1;

SDO_LRS.VALID_LRS_PT(SDO_GEOMETRY(3301,NULL,NULL,SDO_ELEM_INFO_ARRAY
-----
TRUE

SQL> -- Locate the point on Route1 at measure 9, offset 0.
SQL> SELECT SDO_LRS.LOCATE_PT(route_geometry, 9, 0)
2 FROM lrs_routes WHERE route_id = 1;

SDO_LRS.LOCATE_PT(ROUTE_GEOMETRY,9,0) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), S
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
9, 4, 9))

```

Spatial Analysis and Mining

This chapter describes the Oracle Spatial support for spatial analysis and mining in Oracle Data Mining (ODM) applications.

Note: To use the features described in this chapter, you must understand the main concepts and techniques explained in the Oracle Data Mining documentation.

For reference information about spatial analysis and mining functions and procedures, see [Chapter 18](#).

This chapter contains the following major sections:

- [Section 8.1, "Spatial Information and Data Mining Applications"](#)
- [Section 8.2, "Spatial Binning for Detection of Regional Patterns"](#)
- [Section 8.3, "Materializing Spatial Correlation"](#)
- [Section 8.4, "Colocation Mining"](#)
- [Section 8.5, "Spatial Clustering"](#)
- [Section 8.6, "Location Prospecting"](#)

8.1 Spatial Information and Data Mining Applications

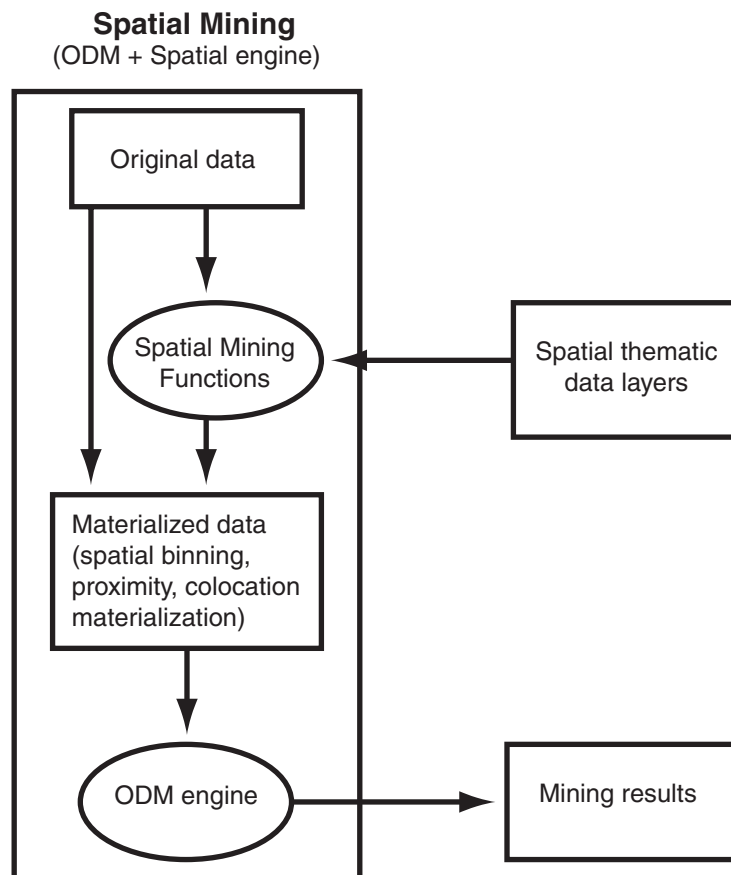
ODM allows automatic discovery of knowledge from a database. Its techniques include discovering hidden associations between different data attributes, classification of data based on some samples, and clustering to identify intrinsic patterns. Effective with Oracle Database 10g, spatial data can be materialized for inclusion in data mining applications. Thus, ODM might enable you to discover that sales prospects with addresses located in specific areas (neighborhoods, cities, or regions) are more likely to watch a particular television program or to respond favorably to a particular advertising solicitation. (The addresses are geocoded into longitude/latitude points and stored in an Oracle Spatial geometry object.)

In many applications, data at a specific location is influenced by data in the neighborhood. For example, the value of a house is largely determined by the value of other houses in the neighborhood. This phenomenon is called *spatial correlation* (or, neighborhood influence), and is discussed further in [Section 8.3](#). The spatial analysis and mining features in Oracle Spatial let you exploit spatial correlation by using the location attributes of data items in several ways: for binning (discretizing) data into regions (such as categorizing data into northern, southern, eastern, and western regions), for materializing the influence of neighborhood (such as number of

customers within a two-mile radius of each store), and for identifying colocated data items (such as video rental stores and pizza restaurants).

To perform spatial data mining, you materialize spatial predicates and relationships for a set of spatial data using thematic layers. Each layer contains data about a specific kind of spatial data (that is, having a specific "theme"), for example, parks and recreation areas, or demographic income data. The spatial materialization could be performed as a preprocessing step before the application of data mining techniques, or it could be performed as an intermediate step in spatial mining, as shown in [Figure 8-1](#).

Figure 8-1 Spatial Mining and Oracle Data Mining



Notes on [Figure 8-1](#):

- The original data, which included spatial and nonspatial data, is processed to produce materialized data.
- Spatial data in the original data is processed by spatial mining functions to produce materialized data. The processing includes such operations as spatial binning, proximity, and colocation materialization.
- The ODM engine processes materialized data (spatial and nonspatial) to generate mining results.

The following are examples of the kinds of data mining applications that could benefit from including spatial information in their processing:

- Business prospecting: Determine if colocation of a business with another franchise (such as colocation of a Pizza Hut restaurant with a Blockbuster video store) might improve its sales.
- Store prospecting: Find a good store location that is within 50 miles of a major city and inside a state with no sales tax. (Although 50 miles is probably too far to drive to avoid a sales tax, many customers may live near the edge of the 50-mile radius and thus be near the state with no sales tax.)
- Hospital prospecting: Identify the best locations for opening new hospitals based on the population of patients who live in each neighborhood.
- Spatial region-based classification or personalization: Determine if southeastern United States customers in a certain age or income category are more likely to prefer "soft" or "hard" rock music.
- Automobile insurance: Given a customer's home or work location, determine if it is in an area with high or low rates of accident claims or auto thefts.
- Property analysis: Use colocation rules to find hidden associations between proximity to a highway and either the price of a house or the sales volume of a store.
- Property assessment: In assessing the value of a house, examine the values of similar houses in a neighborhood, and derive an estimate based on variations and spatial correlation.

8.2 Spatial Binning for Detection of Regional Patterns

Spatial binning (spatial discretization) discretizes the location values into a small number of groups associated with geographical areas. The assignment of a location to a group can be done by any of the following methods:

- Reverse geocoding the longitude/latitude coordinates to obtain an address that specifies (for United States locations) the ZIP code, city, state, and country
- Checking a spatial bin table to determine which bin this specific location belongs in

You can then apply ODM techniques to the discretized locations to identify interesting regional patterns or association rules. For example, you might discover that customers in area A prefer regular soda, while customers in area B prefer diet soda.

The following functions and procedures, documented in [Chapter 18](#), perform operations related to spatial binning:

- [SDO_SAM.BIN_GEOMETRY](#)
- [SDO_SAM.BIN_LAYER](#)

8.3 Materializing Spatial Correlation

Spatial correlation (or, *neighborhood influence*) refers to the phenomenon of the location of a specific object in an area affecting some nonspatial attribute of the object. For example, the value (nonspatial attribute) of a house at a given address (geocoded to give a spatial attribute) is largely determined by the value of other houses in the neighborhood.

To use spatial correlation in a data mining application, you materialize the spatial correlation by adding attributes (columns) in a data mining table. You use associated

thematic tables to add the appropriate attributes. You then perform mining tasks on the data mining table using ODM functions.

The following functions and procedures, documented in [Chapter 18](#), perform operations related to materializing spatial correlation:

- [SDO_SAM.SIMPLIFY_GEOMETRY](#)
- [SDO_SAM.SIMPLIFY_LAYER](#)
- [SDO_SAM.AGGREGATES_FOR_GEOMETRY](#)
- [SDO_SAM.AGGREGATES_FOR_LAYER](#)

8.4 Colocation Mining

Colocation is the presence of two or more spatial objects at the same location or at significantly close distances from each other. Colocation patterns can indicate interesting associations among spatial data objects with respect to their nonspatial attributes. For example, a data mining application could discover that sales at franchises of a specific pizza restaurant chain were higher at restaurants collocated with video stores than at restaurants not collocated with video stores.

Two types of colocation mining are supported:

- Colocation of items in a data mining table. Given a data layer, this approach identifies the colocation of multiple features. For example, predator and prey species could be collocated in animal habitats, and high-sales pizza restaurants could be collocated with high-sales video stores. You can use a reference-feature approach (using one feature as a reference and the other features as thematic attributes, and materializing all neighbors for the reference feature) or a buffer-based approach (materializing all items that are within all windows of a specified size).
- Colocation with thematic layers. Given several data layers, this approach identifies colocation across the layers. For example, given a lakes layer and a vegetation layer, lakes could be collocated with areas of high vegetation. You materialize the data, add categorical and numerical spatial relationships to the data mining table, and apply the ODM Association-Rule mechanisms.

The following functions and procedures, documented in [Chapter 18](#), perform operations related to colocation mining:

- [SDO_SAM.COLOCATED_REFERENCE_FEATURES](#)
- [SDO_SAM.BIN_GEOMETRY](#)

8.5 Spatial Clustering

Spatial clustering returns cluster geometries for a layer of data. An example of spatial clustering is the clustering of crime location data.

The [SDO_SAM.SPATIAL_CLUSTERS](#) function, documented in [Chapter 18](#), performs spatial clustering. This function requires a spatial R-tree index on the geometry column of the layer, and it returns a set of `SDO_REGION` objects where the geometry column specifies the boundary of each cluster and the `geometry_key` value is set to null.

You can use the [SDO_SAM.BIN_GEOMETRY](#) function, with the returned spatial clusters in the bin table, to identify the cluster to which a geometry belongs.

8.6 Location Prospecting

Location prospecting can be performed by using thematic layers to compute aggregates for a layer, and choosing the locations that have the maximum values for computed aggregates.

The following functions, documented in [Chapter 18](#), perform operations related to location prospecting:

- [SDO_SAM.AGGREGATES_FOR_GEOMETRY](#)
- [SDO_SAM.AGGREGATES_FOR_LAYER](#)
- [SDO_SAM.TILED_AGGREGATES](#)

Extending Spatial Indexing Capabilities

This chapter shows how to create and use spatial indexes on objects other than a geometry column. In other chapters, the focus is on indexing and querying spatial data that is stored in a single column of type SDO_GEOMETRY. This chapter shows how to:

- Embed an SDO_GEOMETRY object in a user-defined object type, and index the geometry attribute of that type (see [Section 9.1](#))
- Create and use a function-based index where the function returns an SDO_GEOMETRY object (see [Section 9.2](#))

The techniques in this chapter are intended for experienced and knowledgeable application developers. You should be familiar with the Spatial concepts and techniques described in other chapters. You should also be familiar with, or able to learn about, relevant Oracle database features, such as user-defined data types and function-based indexing.

9.1 SDO_GEOMETRY Objects in User-Defined Type Definitions

The SDO_GEOMETRY type can be embedded in a user-defined data type definition. The procedure is very similar to that for using the SDO_GEOMETRY type for a spatial data column:

1. Create the user-defined data type.
2. Create a table with a column based on that data type.
3. Insert data into the table.
4. Update the USER_SDO_GEOM_METADATA view.
5. Create the spatial index on the geometry attribute.
6. Perform queries on the data.

For example, assume that you want to follow the cola markets scenario in the simplified example in [Section 2.1](#), but want to incorporate the market name attribute and the geometry attribute in a single type. First, create the user-defined data type, as in the following example that creates an object type named MARKET_TYPE:

```
CREATE OR REPLACE TYPE market_type AS OBJECT
  (name VARCHAR2(32), shape SDO_GEOMETRY);
/
```

Create a table that includes a column based on the user-defined type. The following example creates a table named COLA_MARKETS_2 that will contain the same information as the COLA_MARKETS table used in the example in [Section 2.1](#).

```
CREATE TABLE cola_markets_2 (
```

```
mkt_id NUMBER PRIMARY KEY,
market MARKET_TYPE);
```

Insert data into the table, using the object type name as a constructor. For example:

```
INSERT INTO cola_markets_2 VALUES(
  1,
  MARKET_TYPE('cola_a',
    SDO_GEOMETRY(
      2003, -- two-dimensional polygon
      NULL,
      NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
      SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
        -- define rectangle (lower left and upper right)
    )
  )
);
```

Update the USER_SDO_GEOM_METADATA view, using dot-notation to specify the column name and spatial attribute. The following example specifies MARKET.SHAPE as the COLUMN_NAME (explained in [Section 2.6.2](#)) in the metadata view.

```
INSERT INTO user_sdo_geom_metadata
  (TABLE_NAME,
   COLUMN_NAME,
   DIMINFO,
   SRID)
VALUES (
  'cola_markets_2',
  'market.shape',
  SDO_DIM_ARRAY( -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
  ),
  NULL -- SRID
);
```

Create the spatial index, specifying the column name and spatial attribute using dot-notation. For example.

```
CREATE INDEX cola_spatial_idx_2
ON cola_markets_2(market.shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Perform queries on the data, using dot-notation to refer to attributes of the user-defined type. The following simple query returns information associated with the cola market named cola_a.

```
SELECT c.mkt_id, c.market.name, c.market.shape
FROM cola_markets_2 c
WHERE c.market.name = 'cola_a';
```

The following query returns information associated with all geometries that have any spatial interaction with a specified query window, namely, the rectangle with lower-left coordinates (4,6) and upper-right coordinates (8,8).

```
SELECT c.mkt_id, c.market.name, c.market.shape
FROM cola_markets_2 c
WHERE SDO_RELATE(c.market.shape,
  SDO_GEOMETRY(2003, NULL, NULL,
```

```

SDO_ELEM_INFO_ARRAY(1,1003,3),
SDO_ORDINATE_ARRAY(4,6, 8,8)),
'mask=anyinteract' = 'TRUE';

```

9.2 SDO_GEOMETRY Objects in Function-Based Indexes

A function-based spatial index facilitates queries that use locational information (of type SDO_GEOMETRY) returned by a function or expression. In this case, the spatial index is created based on the precomputed values returned by the function or expression.

If you are not already familiar with function-based indexes, see the following for detailed explanations of their benefits, options, and requirements, as well as usage examples:

- *Oracle Database Application Developer's Guide - Fundamentals*
- *Oracle Database Administrator's Guide*

The procedure for using an SDO_GEOMETRY object in a function-based index is as follows:

1. Create the function that returns an SDO_GEOMETRY object.
The function must be declared as DETERMINISTIC.
2. If the spatial data table does not already exist, create it, and insert data into the table.
3. Update the USER_SDO_GEOM_METADATA view.
4. Create the spatial index.
For a function-based spatial index, the number of parameters must not exceed 32.
5. Perform queries on the data.

The rest of this section describes two examples of using function-based indexes. In both examples, a function is created that returns an SDO_GEOMETRY object, and a spatial index is created on that function. In the first example, the input parameters to the function are a standard Oracle data type (NUMBER). In the second example, the input to the function is a user-defined object type.

9.2.1 Example: Function with Standard Types

In the following example, the input parameters to the function used for the function-based index are standard numeric values (longitude and latitude).

Assume that you want to create a function that returns the longitude and latitude of a point and to use that function in a spatial index. First, create the function, as in the following example that creates a function named GET_LONG_LAT_PT:

```

-- Create a function to return a point geometry (SDO_GTYPE = 2001) with
-- input of 2 numbers: longitude and latitude (SDO_SRID = 8307, for
-- "Longitude / Latitude (WGS 84)", probably the most widely used
-- coordinate system, and the one used for GPS devices.
-- Specify DETERMINISTIC for the function.

create or replace function get_long_lat_pt(longitude in number,
                                           latitude in number)
return SDO_GEOMETRY deterministic is
begin
    return sdo_geometry(2001, 8307,

```

```

        sdo_point_type(longitude, latitude, NULL),NULL, NULL);
end;
/

```

If the spatial data table does not already exist, create the table and add data to it, as in the following example that creates a table named LONG_LAT_TABLE:

```

create table LONG_LAT_TABLE
(longitude number, latitude number, name varchar2(32));

insert into LONG_LAT_TABLE values (10,10, 'Place1');
insert into LONG_LAT_TABLE values (20,20, 'Place2');
insert into LONG_LAT_TABLE values (30,30, 'Place3');

```

Update the USER_SDO_GEOM_METADATA view, using dot-notation to specify the schema name and function name. The following example specifies SCOTT.GET_LONG_LAT_PT(LONGITUDE,LATITUDE) as the COLUMN_NAME (explained in [Section 2.6.2](#)) in the metadata view.

```

-- Set up the metadata entry for this table.
-- The column name sets up the function on top
-- of the two columns used in this function,
-- along with the owner of the function.
insert into user_sdo_geom_metadata values('LONG_LAT_TABLE',
'scott.get_long_lat_pt(longitude,latitude)',
sdo_dim_array(
sdo_dim_element('Longitude', -180, 180, 0.005),
sdo_dim_element('Latitude', -90, 90, 0.005)), 8307);

```

Create the spatial index, specifying the function name with parameters. For example:

```

create index LONG_LAT_TABLE_IDX on
LONG_LAT_TABLE(get_long_lat_pt(longitude,latitude))
indextype is mdsys.spatial_index;

```

Perform queries on the data. In the following example, the two queries accomplish the same thing; however, the first query does not use a user-defined function (instead using a constructor to specify the point), whereas the second query uses the function to specify the point.

```

-- First query: call sdo_filter with an SDO_GEOMETRY constructor
select name from LONG_LAT_TABLE a
where sdo_filter(get_long_lat_pt(a.longitude,a.latitude),
sdo_geometry(2001, 8307,
sdo_point_type(10,10,NULL), NULL, NULL)
)='TRUE';

-- Second query: call sdo_filter with the function that returns an sdo_geometry
select name from LONG_LAT_TABLE a
where sdo_filter(get_long_lat_pt(a.longitude,a.latitude),
get_long_lat_pt(10,10)
)='TRUE';

```

9.2.2 Example: Function with a User-Defined Object Type

In the following example, the input parameter to the function used for the function-based index is an object of a user-defined type that includes the longitude and latitude.

Assume that you want to create a function that returns the longitude and latitude of a point and to create a spatial index on that function. First, create the user-defined data

type, as in the following example that creates an object type named LONG_LAT and its member function GetGeometry:

```
create type long_lat as object (
  longitude number,
  latitude number,
member function GetGeometry(SELF in long_lat)
RETURN SDO_GEOMETRY DETERMINISTIC)
/

create or replace type body long_lat as
  member function GetGeometry(self in long_lat)
  return SDO_GEOMETRY is
  begin
    return sdo_geometry(2001, 8307,
      sdo_point_type(longitude, latitude, NULL), NULL,NULL);
  end;
end;
/
```

If the spatial data table does not already exist, create the table and add data to it, as in the following example that creates a table named TEST_LONG_LAT:

```
create table test_long_lat
  (location long_lat, name varchar2(32));

insert into test_long_lat values (long_lat(10,10), 'Place1');
insert into test_long_lat values (long_lat(20,20), 'Place2');
insert into test_long_lat values (long_lat(30,30), 'Place3');
```

Update the USER_SDO_GEOM_METADATA view, using dot-notation to specify the schema name, table name, and function name and parameter value. The following example specifies SCOTT.LONG_LAT.GetGeometry(LOCATION) as the COLUMN_NAME (explained in [Section 2.6.2](#)) in the metadata view.

```
insert into user_sdo_geom_metadata values('test_long_lat',
  'scott.long_lat.GetGeometry(location)',
  sdo_dim_array(
    sdo_dim_element('Longitude', -180, 180, 0.005),
    sdo_dim_element('Latitude', -90, 90, 0.005)), 8307);
```

Create the spatial index, specifying the column name and function name using dot-notation. For example:

```
create index test_long_lat_idx on test_long_lat(location.GetGeometry())
  indextype is mdsys.spatial_index;
```

Perform queries on the data. The following query performs a primary filter operation, asking for the names of geometries that are likely to interact spatially with point (10,10).

```
SELECT a.name FROM test_long_lat a
  WHERE SDO_FILTER(a.location.GetGeometry(),
    SDO_GEOMETRY(2001, 8307,
      SDO_POINT_TYPE(10,10,NULL), NULL, NULL)
    ) = 'TRUE';
```


Part II

Reference Information

This document has three parts:

- [Part I](#) provides conceptual and usage information about Oracle Spatial.
- Part II provides reference information about Oracle Spatial operators, functions, and procedures.
- [Part III](#) provides supplementary information (appendixes and a glossary).

Part II contains the following chapters with reference information:

- [Chapter 10, "SQL Statements for Indexing Spatial Data"](#)
- [Chapter 11, "Spatial Operators"](#)
- [Chapter 12, "Spatial Aggregate Functions"](#)
- [Chapter 13, "SDO_CS Package \(Coordinate System Transformation\)"](#)
- [Chapter 14, "SDO_GCDR Package \(Geocoding\)"](#)
- [Chapter 15, "SDO_GEOM Package \(Geometry\)"](#)
- [Chapter 16, "SDO_LRS Package \(Linear Referencing System\)"](#)
- [Chapter 17, "SDO_MIGRATE Package \(Upgrading\)"](#)
- [Chapter 18, "SDO_SAM Package \(Spatial Analysis and Mining\)"](#)
- [Chapter 19, "SDO_TUNE Package \(Tuning\)"](#)
- [Chapter 20, "SDO_UTIL Package \(Utility\)"](#)

To understand the examples in the reference chapters, you must understand the conceptual and data type information in [Chapter 2, "Spatial Data Types and Metadata"](#), especially [Section 2.2, "SDO_GEOMETRY Object Type"](#).

SQL Statements for Indexing Spatial Data

This chapter describes the SQL statements used when working with the spatial object data type. The statements are listed in [Table 10–1](#).

Table 10–1 *Spatial Index Creation and Usage Statements*

Statement	Description
ALTER INDEX	Alters specific parameters for a spatial index.
ALTER INDEX REBUILD	Rebuilds a spatial index or a specified partition of a partitioned index.
ALTER INDEX RENAME TO	Changes the name of a spatial index or a partition of a spatial index.
CREATE INDEX	Creates a spatial index on a column of type SDO_GEOMETRY.
DROP INDEX	Deletes a spatial index.

This chapter focuses on using these SQL statements with spatial indexes. For complete reference information about any statement, see *Oracle Database SQL Reference*.

Bold italic text is often used in the **Keywords and Parameters** sections in this chapter to identify a grouping of keywords, followed by specific keywords in the group. For example, *INDEX_PARAMS* identifies the start of a group of index-related keywords.

ALTER INDEX

Purpose

Alters specific parameters for a spatial index.

Syntax

```
ALTER INDEX [schema.]index PARAMETERS ('index_params [physical_storage_params]' )
  [{ NOPARALLEL | PARALLEL [ integer ] }];
```

Keywords and Parameters

Value	Description
<i>INDEX_PARAMS</i>	Changes the characteristics of the spatial index.
sdo_indx_dims	Specifies the number of dimensions to be indexed. For example, a value of 2 causes the first two dimensions to be indexed. Must be less than or equal to the number of actual dimensions (number of SDO_DIM_ELEMENT instances in the dimensional array that describes the geometry objects in the column). If the value is 3 or higher, the only Spatial operator that can be used on the indexed geometries is SDO_FILTER ; the other operators described in Chapter 11 cannot be used. Data type is NUMBER. Default = 2.
sdo_rtr_pctfree	Specifies the minimum percentage of slots in each index tree node to be left empty when the index is created. Slots that are left empty can be filled later when new data is inserted into the table. The value can range from 0 to 50. The default value is best for most applications; however, a value of 0 is recommended if no updates will be performed to the geometry column. Data type is NUMBER. Default = 10.
<i>PHYSICAL_STORAGE_PARAMS</i>	Determines the storage parameters used for altering the spatial index data table. A spatial index data table is a standard Oracle table with a prescribed format. Not all physical storage parameters that are allowed in the STORAGE clause of a CREATE TABLE statement are supported. The following is a list of the supported subset.
tablespace	Specifies the tablespace in which the index data table is created. This parameter is the same as TABLESPACE in the STORAGE clause of a CREATE TABLE statement.
initial	Is the same as INITIAL in the STORAGE clause of a CREATE TABLE statement.
next	Is the same as NEXT in the STORAGE clause of a CREATE TABLE statement.
minextents	Is the same as MINEXTENTS in the STORAGE clause of a CREATE TABLE statement.
maxextents	Is the same as MAXEXTENTS in the STORAGE clause of a CREATE TABLE statement.
pctincrease	Is the same as PCTINCREASE in the STORAGE clause of a CREATE TABLE statement.

Value	Description
{ <i>NOPARALLEL</i> <i>PARALLEL</i> [<i>integer</i>] }	Controls whether serial (<i>NOPARALLEL</i>) execution or parallel (<i>PARALLEL</i>) execution is used for subsequent queries and DML operations that use the index. For parallel execution you can specify an integer value of degree of parallelism. See the Usage Notes for the CREATE INDEX statement for guidelines and restrictions that apply to the use of the <i>PARALLEL</i> keyword. Default = <i>NOPARALLEL</i> . (If <i>PARALLEL</i> is specified without an integer value, the Oracle database calculates the optimum degree of parallelism.)

Prerequisites

- You must have EXECUTE privileges on the index type and its implementation type.
- The spatial index to be altered is not marked in-progress.

Usage Notes

Use this statement to change the parameters of an existing index.

See the Usage Notes for the [CREATE INDEX](#) statement for usage information about many of the other available parameters.

Examples

The following example modifies the tablespace for partition IP2 of the spatial index named BGI.

```
ALTER INDEX bgi MODIFY PARTITION ip2
  PARAMETERS ('tablespace=TBS_3');
```

Related Topics

- [ALTER INDEX REBUILD](#)
- [ALTER INDEX RENAME TO](#)
- [CREATE INDEX](#)
- ALTER TABLE (clauses for partition maintenance) in *Oracle Database SQL Reference*

ALTER INDEX REBUILD

Syntax

```
ALTER INDEX [schema.]index REBUILD
  [PARAMETERS ('rebuild_params [physical_storage_params]' ) ]
  [{ NOPARALLEL | PARALLEL [ integer ] }];
```

or

```
ALTER INDEX [schema.]index REBUILD ONLINE
  [PARAMETERS ('rebuild_params [physical_storage_params]' ) ]
  [{ NOPARALLEL | PARALLEL [ integer ] }];
```

or

```
ALTER INDEX [schema.]index REBUILD PARTITION partition
  [PARAMETERS ('rebuild_params [physical_storage_params]' )];
```

Purpose

Rebuilds a spatial index or a specified partition of a partitioned index.

Keywords and Parameters

Value	Description
<i>REBUILD_PARAMS</i>	Specifies in a command string the index parameters to use in rebuilding the spatial index.
index_status=cleanup	For an online rebuild operation (ALTER INDEX REBUILD ONLINE), performs cleanup operations on tables associated with the older version of the index.
layer_gtype	Checks to ensure that all geometries are of a specified geometry type. The value must be from the Geometry Type column of Table 2–1 in Section 2.2.1 (except that UNKNOWN_GEOMETRY is not allowed). In addition, specifying POINT allows for optimized processing of point data. Data type is VARCHAR2.
sdo_dml_batch_size	Specifies the number of index updates to be processed in each batch of updates after a commit operation. The default value is 1000. For example, if you insert 3500 rows into the spatial table and then perform a commit operation, the updates to the spatial index table are performed in four batches of insert operations (1000, 1000, 1000, and 500). See the Usage Notes for the CREATE INDEX statement for more information. Data type is NUMBER. Default = 1000.
sdo_indx_dims	Specifies the number of dimensions to be indexed. For example, a value of 2 causes the first two dimensions to be indexed. Must be less than or equal to the number of actual dimensions (number of SDO_DIM_ELEMENT instances in the dimensional array that describes the geometry objects in the column). If the value is 3 or higher, the only Spatial operator that can be used on the indexed geometries is SDO_FILTER ; the other operators described in Chapter 11 cannot be used. Data type is NUMBER. Default = 2.

Value	Description
sdo_rtr_pctfree	Specifies the minimum percentage of slots in each index tree node to be left empty when the index is created. Slots that are left empty can be filled later when new data is inserted into the table. The value can range from 0 to 50. Data type is NUMBER. Default = 10.
PHYSICAL_STORAGE_PARAMS	Determines the storage parameters used for rebuilding the spatial index data table. A spatial index data table is a regular Oracle table with a prescribed format. Not all physical storage parameters that are allowed in the STORAGE clause of a CREATE TABLE statement are supported. The following is a list of the supported subset.
tablespace	Specifies the tablespace in which the index data table is created. Same as TABLESPACE in the STORAGE clause of a CREATE TABLE statement.
initial	Is the same as INITIAL in the STORAGE clause of a CREATE TABLE statement.
next	Is the same as NEXT in the STORAGE clause of a CREATE TABLE statement.
minextents	Is the same as MINEXTENTS in the STORAGE clause of a CREATE TABLE statement.
maxextents	Is the same as MAXEXTENTS in the STORAGE clause of a CREATE TABLE statement.
pctincrease	Is the same as PCTINCREASE in the STORAGE clause of a CREATE TABLE statement.
{ <i>NOPARALLEL</i> <i>PARALLEL</i> [<i>integer</i>] }	Controls whether serial (<i>NOPARALLEL</i>) execution or parallel (<i>PARALLEL</i>) execution is used for the rebuilding of the index and for subsequent queries and DML operations that use the index. For parallel execution you can specify an integer value of degree of parallelism. See the Usage Notes for the CREATE INDEX statement for guidelines and restrictions that apply to the use of the <i>PARALLEL</i> keyword. Default = <i>NOPARALLEL</i> . (If <i>PARALLEL</i> is specified without an integer value, the Oracle database calculates the optimum degree of parallelism.)

Prerequisites

- You must have EXECUTE privileges on the index type and its implementation type.
- The spatial index to be altered is not marked in-progress.

Usage Notes

An ALTER INDEX REBUILD 'rebuild_params' statement rebuilds the index using supplied parameters. Spatial index creation involves creating and inserting index data, for each row in the underlying table column being spatially indexed, into a table with a prescribed format. All rows in the underlying table are processed before the insertion of index data is committed, and this requires adequate rollback segment space.

The ONLINE keyword rebuilds the index without blocking the index; that is, queries can use the spatial index while it is being rebuilt. However, after all queries issued during the rebuild operation have completed, you must clean up the old index information (in the MDRT tables) by entering a SQL statement in the following form:

```
ALTER INDEX [schema.]index REBUILD ONLINE PARAMETERS ('index_status=cleanup');
```

The following limitations apply to the use of the ONLINE keyword:

- Only query operations are permitted while the index is being rebuilt. Insert, update, and delete operations that would affect the index are blocked while the index is being rebuilt.
- You cannot use the ONLINE keyword for a rebuild operation if the index was created using the 'sdo_non_leaf_tbl=TRUE' parameter.
- You cannot use the ONLINE keyword for a partitioned spatial index.

The ALTER INDEX REBUILD statement does not use any previous parameters from the index creation. All parameters should be specified for the index you want to rebuild.

For more information about using the `layer_gtype` keyword to constrain data in a layer to a geometry type, see [Section 4.1.2](#).

With a partitioned spatial index, you must use a separate ALTER INDEX REBUILD statement for each partition to be rebuilt.

See also the Usage Notes for the [CREATE INDEX](#) statement for usage information about many of the available parameters and about the use of the PARALLEL keyword.

Examples

The following example rebuilds OLDINDEX and specifies the tablespace in which to create the index data table.

```
ALTER INDEX oldindex REBUILD PARAMETERS('tablespace=TBS_3');
```

Related Topics

- [CREATE INDEX](#)
- [DROP INDEX](#)
- ALTER TABLE and ALTER INDEX (clauses for partition maintenance) in *Oracle Database SQL Reference*

ALTER INDEX RENAME TO

Syntax

```
ALTER INDEX [schema.]index RENAME TO <new_index_name>;
```

```
ALTER INDEX [schema.]index PARTITION partition RENAME TO <new_partition_name>;
```

Purpose

Changes the name of a spatial index or a partition of a spatial index.

Keywords and Parameters

Value	Description
new_index_name	Specifies the new name of the index.
new_partition_name	Specifies the new name of the partition.

Prerequisites

- You must have EXECUTE privileges on the index type and its implementation type.
- The spatial index to be altered is not marked in-progress.

Usage Notes

None.

Examples

The following example renames OLDINDEX to NEWINDEX.

```
ALTER INDEX oldindex RENAME TO newindex;
```

Related Topics

- [CREATE INDEX](#)
- [DROP INDEX](#)

CREATE INDEX

Syntax

```
CREATE INDEX [schema.]index ON [schema.]table (column)
    INDEXTYPE IS MDSYS.SPATIAL_INDEX
    [PARAMETERS ('index_params [physical_storage_params]' )]
    [{ NOPARALLEL | PARALLEL [ integer ] }];
```

Purpose

Creates a spatial index on a column of type SDO_GEOMETRY.

Keywords and Parameters

Value	Description
<i>INDEX_PARAMS</i>	Determines the characteristics of the spatial index.
geodetic	'geodetic=FALSE' allows a non-geodetic index to be built on geodetic data, but with restrictions. (FALSE is the only acceptable value for this keyword.) See the Usage Notes for more information. Data type is VARCHAR2.
layer_gtype	Checks to ensure that all geometries are of a specified geometry type. The value must be from the Geometry Type column of Table 2–1 in Section 2.2.1 (except that UNKNOWN_GEOMETRY is not allowed). In addition, specifying POINT allows for optimized processing of point data. Data type is VARCHAR2.
sdo_dml_batch_size	Specifies the number of index updates to be processed in each batch of updates after a commit operation. The default value is 1000. For example, if you insert 3500 rows into the spatial table and then perform a commit operation, the updates to the spatial index table are performed in four batches of insert operations (1000, 1000, 1000, and 500). See the Usage Notes for more information. Data type is NUMBER. Default = 1000.
sdo_indx_dims	Specifies the number of dimensions to be indexed. For example, a value of 2 causes the first two dimensions to be indexed. Must be less than or equal to the number of actual dimensions (number of SDO_DIM_ELEMENT instances in the dimensional array that describes the geometry objects in the column). If the value is 3 or higher, the only Spatial operator that can be used on the indexed geometries is SDO_FILTER ; the other operators described in Chapter 11 cannot be used. Data type is NUMBER. Default = 2.
sdo_non_leaf_tbl	'sdo_non_leaf_tbl=TRUE' creates a separate index table (with a name in the form MDNT_...\$) for nonleaf nodes of the index, in addition to creating an index table (with a name in the form MDRT_...\$) for leaf nodes. 'sdo_non_leaf_tbl=FALSE' creates a single table (with a name in the form MDRT_...\$) for both leaf nodes and nonleaf nodes of the index. See the Usage Notes for more information. Data type is VARCHAR2. Default = FALSE

Value	Description
<code>sdo_rtr_pctfree</code>	Specifies the minimum percentage of slots in each index tree node to be left empty when the index is created. Slots that are left empty can be filled later when new data is inserted into the table. The value can range from 0 to 50. Data type is NUMBER. Default = 10.
<i>PHYSICAL_STORAGE_PARAMS</i>	Determines the storage parameters used for creating the spatial index data table. A spatial index data table is a regular Oracle table with a prescribed format. Not all physical storage parameters that are allowed in the STORAGE clause of a CREATE TABLE statement are supported. The following is a list of the supported subset.
<code>tablespace</code>	Specifies the tablespace in which the index data table is created. Same as TABLESPACE in the STORAGE clause of a CREATE TABLE statement.
<code>initial</code>	Is the same as INITIAL in the STORAGE clause of a CREATE TABLE statement.
<code>next</code>	Is the same as NEXT in the STORAGE clause of a CREATE TABLE statement.
<code>minextents</code>	Is the same as MINEXTENTS in the STORAGE clause of a CREATE TABLE statement.
<code>maxextents</code>	Is the same as MAXEXTENTS in the STORAGE clause of a CREATE TABLE statement.
<code>pctincrease</code>	Is the same as PCTINCREASE in the STORAGE clause of a CREATE TABLE statement.
<code>work_tablespace</code>	Specifies the tablespace for temporary tables used in creating the index. (Applies only to creating spatial R-tree indexes, and not to other types of indexes.)
<code>{ NOPARALLEL PARALLEL [integer] }</code>	Controls whether serial (NOPARALLEL) execution or parallel (PARALLEL) execution is used for the creation of the index and for subsequent queries and DML operations that use the index. For parallel execution you can specify an integer value of degree of parallelism. See the Usage Notes for more information about parallel index creation. Default = NOPARALLEL. (If PARALLEL is specified without an integer value, the Oracle database calculates the optimum degree of parallelism.)

Prerequisites

- All current SQL CREATE INDEX prerequisites apply.
- You must have EXECUTE privilege on the index type and its implementation type.
- The USER_SDO_GEOM_METADATA view must contain an entry with the dimensions and coordinate boundary information for the table column to be spatially indexed.

Usage Notes

For information about spatial indexes, see [Section 1.7](#).

Before you create a spatial index, be sure that the rollback segment size and the SORT_AREA_SIZE parameter value are adequate, as described in [Section 4.1](#).

If an R-tree index is used on linear referencing system (LRS) data and if the LRS data has four dimensions (three plus the M dimension), the `sdo_indx_dims` parameter

must be used and must specify 3 (the number of dimensions minus one), to avoid the default `sdo_indx_dims` value of 2, which would index only the X and Y dimensions. For example, if the dimensions are X, Y, Z, and M, specify `sdo_indx_dims=3` to index the X, Y, and Z dimensions, but not the measure (M) dimension. (The LRS data model, including the measure dimension, is explained in [Section 7.2](#).)

A partitioned spatial index can be created on a partitioned table. See [Section 4.1.4](#) for more information about partitioned spatial indexes, including benefits and restrictions.

A spatial index cannot be created on an index-organized table.

You can specify the `PARALLEL` keyword to cause the index creation to be parallelized. For example:

```
CREATE INDEX cola_spatial_idx ON cola_markets(shape)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX PARALLEL;
```

For information about using the `PARALLEL` keyword, see the description of the `parallel_clause` in the section on the `CREATE INDEX` statement in *Oracle Database SQL Reference*. In addition, the following notes apply to the use of the `PARALLEL` keyword for creating or rebuilding (using the [ALTER INDEX REBUILD](#) statement) spatial indexes:

- The performance cost and benefits from parallel execution for creating or rebuilding an index depend on system resources and load. If the CPUs or disk controllers are already heavily loaded, you should not specify the `PARALLEL` keyword.
- Specifying `PARALLEL` for creating or rebuilding an index on tables with simple geometries, such as point data, usually results in less performance improvement than on tables with complex geometries.

Other options available for regular indexes (such as `ASC` and `DESC`) are not applicable for spatial indexes.

Spatial index creation involves creating and inserting index data, for each row in the underlying table column being spatially indexed, into a table with a prescribed format. All rows in the underlying table are processed before the insertion of index data is committed, and this requires adequate rollback segment space.

If a tablespace name is provided in the parameters clause, the user (underlying table owner) must have appropriate privileges for that tablespace.

For more information about using the `layer_gtype` keyword to constrain data in a layer to a geometry type, see [Section 4.1.2](#).

The `'geodetic=FALSE'` parameter is not recommended, because much of the Oracle Spatial geodetic support will be disabled. This parameter should only be used if you cannot yet reindex the data. (For more information about geodetic and non-geodetic indexes, see [Section 4.1.1](#).)

Moreover, if you specify `'geodetic=FALSE'`, ensure that the tolerance value stored in the `USER_SDO_GEOM_METADATA` view is what would be used for Cartesian data. That is, do not use meters for the units of the tolerance value, but instead use the number of decimal places in the data followed by a 5 (for example, 0.00005). This tolerance value will be used for spatial operators. When you use spatial functions that require a tolerance value with this data, use the function format that lets you specify a tolerance value, and specify the tolerance value in meters.

The `sdo_dml_batch_size` parameter can improve application performance, because Spatial can preallocate system resources to perform multiple index updates more efficiently than successive single index updates; however, to gain the performance

benefit, you must not perform commit operations after each insert operation or at intervals less than or equal to the `sdo_dml_batch_size` value. You should not specify a value greater than 10000 (ten thousand), because the cost of the additional memory and other resources required will probably outweigh any marginal performance increase resulting from such a value.

Specifying `'sdo_non_leaf_tbl=TRUE'` can help query performance with large data sets if the entire R-tree table may not fit in the KEEP buffer pool. In this case, you must also cause Oracle to buffer the MDNT_...\$ table in the KEEP buffer pool, for example, by using ALTER TABLE and specifying STORAGE (BUFFER_POOL KEEP). For partitioned indexes, the same `sdo_non_leaf_tbl` value must be used for all partitions. Any physical storage parameters, except for `tablespace`, are applied only to the MDRT_...\$ table. The MDNT_...\$ table uses only the `tablespace` parameter, if specified, and default values for all other physical storage parameters.

If you are creating a function-based spatial index, the number of parameters must not exceed 32. For information about using function-based spatial indexes, see [Section 9.2](#).

To determine if a [CREATE INDEX](#) statement for a spatial index has failed, check to see if the DOMIDX_OPSTATUS column in the USER_INDEXES view is set to FAILED. This is different from the case of regular indexes, where you check to see if the STATUS column in the USER_INDEXES view is set to FAILED.

If the [CREATE INDEX](#) statement fails because of an invalid geometry, the ROWID of the failed geometry is returned in an error message along with the reason for the failure.

If the [CREATE INDEX](#) statement fails for any reason, then the [DROP INDEX](#) statement must be used to clean up the partially built index and associated metadata. If [DROP INDEX](#) does not work, add the FORCE parameter and try again.

Examples

The following example creates a spatial R-tree index named COLA_SPATIAL_IDX.

```
CREATE INDEX cola_spatial_idx ON cola_markets(shape)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Related Topics

- [ALTER INDEX](#)
- [DROP INDEX](#)

DROP INDEX

Syntax

```
DROP INDEX [schema.]index [FORCE];
```

Purpose

Deletes a spatial index.

Keywords and Parameters

Value	Description
FORCE	Causes the spatial index to be deleted from the system tables even if the index is marked in-progress or some other error condition occurs.

Prerequisites

You must have EXECUTE privileges on the index type and its implementation type.

Usage Notes

Use [DROP INDEX](#) indexname FORCE to clean up after a failure in the [CREATE INDEX](#) statement.

Examples

The following example deletes a spatial index named OLDINDEX and forces the deletion to be performed even if the index is marked in-process or an error occurs.

```
DROP INDEX oldindex FORCE;
```

Related Topics

- [CREATE INDEX](#)

Spatial Operators

This chapter describes the operators that you can use when working with the spatial object data type. For an overview of spatial operators, including how they differ from spatial procedures and functions, see [Section 1.9](#). [Table 11–1](#) lists the main operators.

Table 11–1 *Main Spatial Operators*

Operator	Description
SDO_FILTER	Specifies which geometries may interact with a given geometry.
SDO_JOIN	Performs a spatial join based on one or more topological relationships.
SDO_NN	Determines the nearest neighbor geometries to a geometry.
SDO_NN_DISTANCE	Returns the distance of an object returned by the SDO_NN operator.
SDO_RELATE	Determines whether or not two geometries interact in a specified way. (See also Table 11–2 for convenient alternative operators for performing specific mask value operations.)
SDO_WITHIN_DISTANCE	Determines if two geometries are within a specified distance from one another.

[Table 11–2](#) lists operators, provided for convenience, that perform an [SDO_RELATE](#) operation of a specific mask type.

Table 11–2 *Convenience Operators for SDO_RELATE Operations*

Operator	Description
SDO_ANYINTERACT	Checks if any geometries in a table have the ANYINTERACT topological relationship with a specified geometry.
SDO_CONTAINS	Checks if any geometries in a table have the CONTAINS topological relationship with a specified geometry.
SDO_COVEREDBY	Checks if any geometries in a table have the COVEREDBY topological relationship with a specified geometry.
SDO_COVERS	Checks if any geometries in a table have the COVERS topological relationship with a specified geometry.
SDO_EQUAL	Checks if any geometries in a table have the EQUAL topological relationship with a specified geometry.
SDO_INSIDE	Checks if any geometries in a table have the INSIDE topological relationship with a specified geometry.

Table 11–2 (Cont.) Convenience Operators for SDO_RELATE Operations

Operator	Description
<code>SDO_ON</code>	Checks if any geometries in a table have the ON topological relationship with a specified geometry.
<code>SDO_OVERLAPBDYDISJOINT</code>	Checks if any geometries in a table have the OVERLAPBDYDISJOINT topological relationship with a specified geometry.
<code>SDO_OVERLAPBDYINTERSECT</code>	Checks if any geometries in a table have the OVERLAPBDYINTERSECT topological relationship with a specified geometry.
<code>SDO_OVERLAPS</code>	Checks if any geometries in a table overlap (that is, have the OVERLAPBDYDISJOINT or OVERLAPBDYINTERSECT topological relationship with) a specified geometry.
<code>SDO_TOUCH</code>	Checks if any geometries in a table have the TOUCH topological relationship with a specified geometry.

The rest of this chapter provides reference information on the operators, listed in alphabetical order.

For information about using operators with topologies, see *Oracle Spatial Topology and Network Data Models*.

SDO_ANYINTERACT

Format

```
SDO_ANYINTERACT(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the ANYINTERACT topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=ANYINTERACT'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_ANYINTERACT(geometry1,geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the ANYINTERACT topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the ANYINTERACT relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 4,6, 8,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).)

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_ANYINTERACT(c.shape,
                        SDO_GEOMETRY(2003, NULL, NULL,
                                      SDO_ELEM_INFO_ARRAY(1,1003,3),
                                      SDO_ORDINATE_ARRAY(4,6, 8,8))
                        ) = 'TRUE';

MKT_ID NAME
-----
      2 cola_b
```

```
1 cola_a  
4 cola_d
```

SDO_CONTAINS

Format

```
SDO_CONTAINS(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the CONTAINS topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=CONTAINS'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_CONTAINS(geometry1, geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the CONTAINS topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the CONTAINS relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 2,2, 4,6). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, only `cola_a` contains the query window geometry.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_CONTAINS(c.shape,
                    SDO_GEOMETRY(2003, NULL, NULL,
                                   SDO_ELEM_INFO_ARRAY(1,1003,3),
                                   SDO_ORDINATE_ARRAY(2,2, 4,6))
                    ) = 'TRUE';

MKT_ID NAME
-----
1 cola_a
```

SDO_COVEREDBY

Format

```
SDO_COVEREDBY(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the COVEREDBY topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=COVEREDBY'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_COVEREDBY(geometry1,geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the COVEREDBY topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the COVEREDBY relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 1,1, 5,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, only cola_a is covered by the query window geometry.

```
SELECT c.mkt_id, c.name
FROM cola_markets c
WHERE SDO_COVEREDBY(c.shape,
    SDO_GEOMETRY(2003, NULL, NULL,
        SDO_ELEM_INFO_ARRAY(1,1003,3),
        SDO_ORDINATE_ARRAY(1,1, 5,8))
    ) = 'TRUE';

MKT_ID NAME
-----
1 cola_a
```

SDO_COVERS

Format

```
SDO_COVERS(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the COVERS topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=COVERS'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_COVERS(geometry1, geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the COVERS topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the COVERS relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 1,1, 4,6). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, only `cola_a` covers the query window geometry.

```
SELECT c.mkt_id, c.name
FROM cola_markets c
WHERE SDO_COVERS(c.shape,
                 SDO_GEOMETRY(2003, NULL, NULL,
                               SDO_ELEM_INFO_ARRAY(1,1003,3),
                               SDO_ORDINATE_ARRAY(1,1, 4,6))
                 ) = 'TRUE';

MKT_ID NAME
-----
1 cola_a
```

SDO_EQUAL

Format

```
SDO_EQUAL(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the EQUAL topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=EQUAL'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_EQUAL(geometry1,geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the EQUAL topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the EQUAL relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 1,1, 5,7). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2–1](#).) In this example, `cola_a` (and only `cola_a`) has the same boundary and interior as the query window geometry.

```
SELECT c.mkt_id, c.name
  FROM cola_markets c
 WHERE SDO_EQUAL(c.shape,
                SDO_GEOMETRY(2003, NULL, NULL,
                SDO_ELEM_INFO_ARRAY(1,1003,3),
                SDO_ORDINATE_ARRAY(1,1, 5,7))
                ) = 'TRUE';

MKT_ID NAME
-----
1 cola_a
```

SDO_FILTER

Format

SDO_FILTER(geometry1, geometry2, param);

Description

Uses the spatial index to identify either the set of spatial objects that are likely to interact spatially with a given object (such as an area of interest), or pairs of spatial objects that are likely to interact spatially. Objects interact spatially if they are not disjoint.

This operator performs only a primary filter operation. The secondary filtering operation, performed by the [SDO_RELATE](#) operator, can be used to determine with certainty if objects interact spatially.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.
param	Optionally specifies either or both of the <code>min_resolution</code> and <code>max_resolution</code> keywords. Data type is VARCHAR2. The <code>min_resolution</code> keyword includes only geometries for which at least one side of the geometry's MBR is equal to or greater than the specified value. For example, <code>min_resolution=10</code> includes only geometries for which the width or the height (or both) of the geometry's MBR is at least 10. (This keyword can be used to exclude geometries that are too small to be of interest.) The <code>max_resolution</code> keyword includes only geometries for which at least one side of the geometry's MBR is less than or equal to the specified value. For example, <code>max_resolution=10</code> includes only geometries for which the width or the height (or both) of the geometry's MBR is less than or equal to 10. (This keyword can be used to exclude geometries that are too large to be of interest.)

Returns

The expression `SDO_FILTER(geometry1,geometry2) = 'TRUE'` evaluates to TRUE for object pairs that are non-disjoint, and FALSE otherwise.

Usage Notes

SDO_FILTER is the only operator that can be used with data that is indexed using more than two dimensions. The operator considers all dimensions specified in the spatial index.

The operator must always be used in a WHERE clause and the condition that includes the operator should be an expression of the form `SDO_FILTER(arg1, arg2) = 'TRUE'`.

`geometry2` can come from a table or be a transient `SDO_GEOMETRY` object (such as a bind variable or `SDO_GEOMETRY` constructor).

- If the `geometry2` column is not spatially indexed, the operator indexes the query window in memory and performance is very good.
- If two or more geometries from `geometry2` are passed to the operator, the `ORDERED` optimizer hint must be specified, and the table in `geometry2` must be specified first in the `FROM` clause.

If `geometry1` and `geometry2` are based on different coordinate systems, `geometry2` is temporarily transformed to the coordinate system of `geometry1` for the operation to be performed, as described in [Section 6.9.1](#).

RLS Restriction: If the `DBMS_RLS.ADD_POLICY` procedure has been used to add a fine-grained access control policy to a table or view, and if the specified policy function uses a spatial operator, the operator must be `SDO_FILTER`. No other spatial operators are supported in that context.

In previous releases, the `SDO_FILTER` operator required a third parameter. Effective with Oracle Spatial release 10.1, the operator has only two parameters. For backward compatibility, any keywords for the third parameter that were supported in the previous release will still work; however, the use of those keywords is discouraged and is not supported for new uses of the operator.

Examples

The following example selects the geometries that are likely to interact with a query window (here, a rectangle with lower-left, upper-right coordinates 4,6, 8,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).)

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_FILTER(c.shape,
    SDO_GEOMETRY(2003, NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(4,6, 8,8))
    ) = 'TRUE';
```

```

MKT_ID NAME
-----
      2 cola_b
      1 cola_a
      4 cola_d
```

The following example is the same as the preceding example, except that it includes only geometries where at least one side of the geometry's MBR is equal to or greater than 4.1. In this case, only `cola_a` and `cola_b` are returned, because their MBRs have at least one side with a length greater than or equal to 4.1. The circle `cola_d` is excluded, because its MBR is a square whose sides have a length of 4.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_FILTER(c.shape,
    SDO_GEOMETRY(2003, NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(4,6, 8,8)),
    4.1);
```

```
'min_resolution=4.1'
) = 'TRUE';

MKT_ID NAME
-----
      2 cola_b
      1 cola_a
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column objects are likely to interact spatially with the GEOMETRY column object in the QUERY_POLYS table that has a GID value of 1.

```
SELECT A.gid
FROM Polygons A, query_polys B
WHERE B.gid = 1
AND SDO_FILTER(A.Geometry, B.Geometry) = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column object is likely to interact spatially with the geometry stored in the aGeom variable.

```
Select A.Gid
FROM Polygons A
WHERE SDO_FILTER(A.Geometry, :aGeom) = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column object is likely to interact spatially with the specified rectangle having the lower-left coordinates (x1,y1) and the upper-right coordinates (x2, y2).

```
Select A.Gid
FROM Polygons A
WHERE SDO_FILTER(A.Geometry, sdo_geometry(2003,NULL,NULL,
                                         sdo_elem_info_array(1,1003,3),
                                         sdo_ordinate_array(x1,y1,x2,y2))
) = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column object is likely to interact spatially with any GEOMETRY column object in the QUERY_POLYS table. In this example, the ORDERED optimizer hint is used and the QUERY_POLYS(geometry2) table is specified first in the FROM clause, because multiple geometries from geometry2 are involved (see the Usage Notes).

```
SELECT /*+ ORDERED */
A.gid
FROM query_polys B, polygons A
WHERE SDO_FILTER(A.Geometry, B.Geometry) = 'TRUE';
```

Related Topics

- [SDO_RELATE](#)

SDO_INSIDE

Format

```
SDO_INSIDE(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the **INSIDE** topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=INSIDE'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_INSIDE(geometry1,geometry2) = 'TRUE'` evaluates to **TRUE** for object pairs that have the **INSIDE** topological relationship, and **FALSE** otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the **INSIDE** relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 5,6, 12,12). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, only `cola_d` (the circle) is inside the query window geometry.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_INSIDE(c.shape,
                  SDO_GEOMETRY(2003, NULL, NULL,
                               SDO_ELEM_INFO_ARRAY(1,1003,3),
                               SDO_ORDINATE_ARRAY(5,6, 12,12))
                  ) = 'TRUE';

MKT_ID NAME
-----
      4 cola_d
```

SDO_JOIN

Format

```
SDO_JOIN(table_name1, column_name1, table_name2, column_name2, params,
preserve_join_order) RETURN SDO_ROWIDSET;
```

Description

Performs a spatial join based on one or more topological relationships.

Keywords and Parameters

Value	Description
table_name1	Name of the first table to be used in the spatial join operation. The table must have a column of type SDO_GEOMETRY. Data type is VARCHAR2.
column_name1	Name of the spatial column of type SDO_GEOMETRY in table_name1. A spatial R-tree index must be defined on this column. Data type is VARCHAR2.
table_name2	Name of the second table to be used in the spatial join operation. (It can be the same as or different from table_name1.) The table must have a column of type SDO_GEOMETRY. Data type is VARCHAR2.
column_name2	Name of the spatial column of type SDO_GEOMETRY in table_name2. A spatial R-tree index must be defined on this column. Data type is VARCHAR2.
params	Optional parameter string of keywords and values; available only if mask=ANYINTERACT. Determines the behavior of the operator. See Table 11-3 in the Usage Notes for information about the available keywords. Data type is VARCHAR2. Default is NULL.
preserve_join_order	Optional parameter to specify if the join order is guaranteed to be preserved during processing of the operator. If the value is 0 (the default), the order of the tables might be changed; if the value is 1, the order of the tables is not changed. Data type is NUMBER. Default is 0.

Returns

SDO_JOIN returns an object of SDO_ROWIDSET, which consists of a table of objects of SDO_ROWIDPAIR. Oracle Spatial defines the type SDO_ROWIDSET as:

```
CREATE TYPE sdo_rowidset as TABLE OF sdo_rowidpair;
```

Oracle Spatial defines the object type SDO_ROWIDPAIR as:

```
CREATE TYPE sdo_rowidpair AS OBJECT
  (rowid1 VARCHAR2(24),
   rowid2 VARCHAR2(24));
```

In the SDO_ROWIDPAIR definition, rowid1 refers to a rowid from table_name1, and rowid2 refers to a rowid from table_name2.

Usage Notes

SDO_JOIN is technically not an operator, but a table function. (For an explanation of table functions, see *PL/SQL User's Guide and Reference*.) However, it is presented in the chapter with Spatial operators because its usage is similar to that of the operators, and because it is not part of a package with other functions and procedures.

This table function is recommended when you need to perform full table joins.

The geometries in `column_name1` and `column_name2` must have the same SRID (coordinate system) value and the same number of dimensions.

For best performance, use the `/*+ ORDERED */` optimizer hint, and specify the SDO_JOIN table function first in the FROM clause.

If a table is version-enabled (using the Workspace Manager feature), you must specify the `<table_name>_LT` table created by Workspace Manager. For example, if the COLA_MARKETS table is version-enabled and you want to perform a spatial join operation on that table, specify COLA_MARKETS_LT (not COLA_MARKETS) with the SDO_JOIN table function. (However, for all other Spatial functions, procedures, and operators, do not use the `<table_name>_LT` name.)

The SDO_JOIN table function is not supported on partitioned tables.

Table 11-3 shows the keywords for the `params` parameter.

Table 11-3 *params* Keywords for the SDO_JOIN Operator

Keyword	Description
mask	<p>The topological relationship of interest. Valid values are 'mask=<value>' where <value> is one or more of the mask values valid for the SDO_RELATE operator (TOUCH, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, EQUAL, INSIDE, COVEREDBY, CONTAINS, COVERS, ANYINTERACT, ON), or FILTER, which checks if the MBRs (the filter-level approximations) intersect. Multiple masks are combined with the logical Boolean operator OR (for example, 'mask=inside+touch'); however, FILTER cannot be combined with any other mask.</p> <p>If this parameter is null or contains an empty string, mask=FILTER is assumed.</p>
distance	<p>Specifies a numeric distance value that is added to the tolerance value (explained in Section 1.5.5) before the relationship checks are performed. For example, if the tolerance is 10 meters and you specify 'distance=100 unit=meter', two objects are considered to have spatial interaction if they are within 110 meters of each other.</p> <p>If you specify distance but not unit, the unit of measurement associated with the data is assumed.</p>
unit	<p>Specifies a unit of measurement to be associated with the distance value (for example, 'distance=100 unit=meter'). See Section 2.8 for more information about unit of measurement specification. If you specify unit, you must also specify distance.</p> <p>Data type is VARCHAR2. Default = unit of measurement associated with the data. For geodetic data, the default is meters.</p>

Examples

The following example joins the COLA_MARKETS table with itself to find, for each geometry, all other geometries that have any spatial interaction with it. (The example uses the definitions and data from [Section 2.1](#).) In this example, rowid1 and rowid2 correspond to the names of the attributes in the SDO_ROWIDPAIR type definition. Note that in the output, cola_d (the circle in [Figure 2-1](#)) interacts only with itself, and not with any of the other geometries.

```
SELECT /*+ ordered */ a.name, b.name
  FROM TABLE(SDO_JOIN('COLA_MARKETS', 'SHAPE',
                      'COLA_MARKETS', 'SHAPE',
                      'mask=ANYINTERACT')) c,
         cola_markets a,
         cola_markets b
 WHERE c.rowid1 = a.rowid AND c.rowid2 = b.rowid
 ORDER BY a.name;
```

NAME	NAME
cola_a	cola_c
cola_a	cola_b
cola_a	cola_a
cola_b	cola_c
cola_b	cola_b
cola_b	cola_a
cola_c	cola_c
cola_c	cola_b
cola_c	cola_a
cola_d	cola_d

10 rows selected.

Related Topics

- [SDO_RELATE](#)

SDO_NN

Format

SDO_NN(geometry1, geometry2, param [, number]);

Description

Uses the spatial index to identify the nearest neighbors for a geometry.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. The nearest neighbor or neighbors to geometry2 will be returned from geometry1. (geometry2 is specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.
param	Determines the behavior of the operator. The available keywords are listed in Table 11-4 . If you do not specify this parameter, the operator returns all rows in increasing distance order from geometry2. Data type is VARCHAR2.
number	If the SDO_NN_DISTANCE ancillary operator is included in the call to SDO_NN, specifies the same number used in the call to SDO_NN_DISTANCE . Data type is NUMBER.

[Table 11-4](#) lists the keywords for the param parameter.

Table 11-4 Keywords for the SDO_NN Param Parameter

Keyword	Description
sdo_batch_size	Specifies the number of rows to be evaluated at a time when the SDO_NN expression may need to be evaluated multiple times in order to return the desired number of results that satisfy the WHERE clause. Available only when an R-tree index is used. If you specify sdo_batch_size=0 (or if you omit the param parameter completely), Spatial calculates a batch size suited to the result set size. See the Usage Notes and Examples for more information. Data type is NUMBER. For example: 'sdo_batch_size=10'
sdo_num_res	If sdo_batch_size is not specified, specifies the number of results (nearest neighbors) to be returned. If sdo_batch_size is specified, this keyword is ignored; instead, use the ROWNUM pseudocolumn to limit the number of results. See the Usage Notes and Examples for more information. Data type is NUMBER. For example: 'sdo_num_res=5'

Table 11–4 (Cont.) Keywords for the SDO_NN Param Parameter

Keyword	Description
unit	<p>If the SDO_NN_DISTANCE ancillary operator is included in the call to SDO_NN, specifies the unit of measurement: a quoted string with <code>unit=</code> and an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table. See Section 2.8 for more information about unit of measurement specification. Data type is VARCHAR2. Default = unit of measurement associated with the data. For geodetic data, the default is meters.</p> <p>For example: 'unit=KM'</p>

Returns

This operator returns the `sdo_num_res` number of objects from `geometry1` that are nearest to `geometry2` in the query. In determining how near two geometry objects are, the shortest possible distance between any two points on the surface of each object is used.

Usage Notes

The operator is disabled if the table does not have a spatial index or if the index has been built on more than two dimensions.

The operator must always be used in a WHERE clause, and the condition that includes the operator should be an expression of the form `SDO_NN(arg1, arg2, '<some_parameter>') = 'TRUE'`.

The operator can be used in two ways:

- If all geometries in the layer are candidates, use the `sdo_num_res` keyword to specify the number of geometries returned.
- If any geometries in the table might be nearer than the geometries specified in the WHERE clause, use the `sdo_batch_size` keyword and use the WHERE clause (including the ROWNUM pseudocolumn) to limit the number of geometries returned.

As an example of the `sdo_batch_size` keyword, assume that a RESTAURANTS table contains different types of restaurants, and you want to find the two nearest Italian restaurants to your hotel. The query might look like the following:

```
SELECT r.name FROM restaurants r WHERE
  SDO_NN(r.geometry, :my_hotel, 'sdo_batch_size=10') = 'TRUE'
  AND r.cuisine = 'Italian' AND ROWNUM <=2;
```

In this example, the `ROWNUM <=2` clause is necessary to limit the number of results returned to no more than 2 where CUISINE is Italian. However, if the `sdo_batch_size` keyword is not specified in this example, and if `sdo_num_res=2` is specified instead of `ROWNUM <=2`, only the two nearest restaurants are considered, regardless of their CUISINE value; and if the CUISINE value of these two rows is not Italian, the query may return no rows.

The `sdo_batch_size` value can affect the performance of nearest neighbor queries. A good general guideline is to specify the number of candidate rows likely to satisfy the WHERE clause. Using the preceding example of a query for Italian restaurants, if approximately 20 percent of the restaurants nearest to the hotel are Italian and if you want 2 restaurants, an `sdo_batch_size` value of 10 will probably result in the best performance. On the other hand, if only approximately 5 percent of the restaurants nearest to the hotel are Italian and if you want 2 restaurants, an `sdo_batch_size` value of 40 would be better.

You can specify `sdo_batch_size=0`, which causes Spatial to calculate a batch size that is suitable for the result set size. However, the calculated batch size may not be optimal, and the calculation incurs some processing overhead; if you can determine a good `sdo_batch_size` value for a query, the performance will probably be better than if you specify `sdo_batch_size=0`.

If the `sdo_batch_size` keyword is specified, any `sdo_num_res` value is ignored. Do not specify both keywords.

Specify the number parameter only if you are using the [SDO_NN_DISTANCE](#) ancillary operator in the call to `SDO_NN`. See the information about the [SDO_NN_DISTANCE](#) operator in this chapter.

If this operator is used with geodetic data, the data must be indexed with an R-tree spatial index. If this operator is used with geodetic data and if the R-tree spatial index is created with `'geodetic=false'` specified, you cannot use the `unit` parameter.

If two or more objects from `geometry1` are an equal distance from `geometry2`, any of the objects can be returned on any call to the function. For example, if `item_a`, `item_b`, and `item_c` are nearest to and equally distant from `geometry2`, and if `sdo_num_res=2`, two of those three objects are returned, but they can be any two of the three.

If the `SDO_NN` operator uses a partitioned spatial index (see [Section 4.1.4](#)), the requested number of geometries is returned for *each* partition that contains candidate rows based on the query criteria. For example, if you request the 5 nearest restaurants to a point and the spatial index has 4 partitions, the operator returns up to 20 (5*4) geometries. In this case, you must use the `ROWNUM` pseudocolumn (here, `WHERE ROWNUM <=5`) to return the 5 nearest restaurants.

If `geometry1` and `geometry2` are based on different coordinate systems, `geometry2` is temporarily transformed to the coordinate system of `geometry1` for the operation to be performed, as described in [Section 6.9.1](#).

`SDO_NN` is not supported for spatial joins.

In some situations the `SDO_NN` operator will not use the spatial index unless an optimizer hint forces the index to be used. This can occur when a query involves a join; and if the optimizer hint is not used in such situations, an internal error occurs. To prevent such errors, you should always specify an optimizer hint to use the spatial index with the `SDO_NN` operator, regardless of how simple or complex the query is. For example, the following excerpt from a query specifies to use the `COLA_SPATIAL_IDX` index that is defined on the `COLA_MARKETS` table:

```
SELECT /*+ INDEX(c cola_spatial_idx) */
      c.mkt_id, c.name, ... FROM cola_markets c, ...;
```

However, if the column predicate in the `WHERE` clause specifies any nonspatial column in the table for `geometry1` that has an associated index, be sure that this index is not used by specifying the `NO_INDEX` hint for that index. For example, if there was an index named `COLA_NAME_IDX` defined on the `NAME` column, you would need to specify the hints in the preceding example as follows:

```
SELECT /*+ INDEX(c cola_spatial_idx) NO_INDEX(c cola_name_idx) */
      c.mkt_id, c.name, ... FROM cola_markets c, ...;
```

(Note, however, that there is no index named `COLA_NAME_IDX` in the example in [Section 2.1](#).)

If you join two or more tables with the SDO_NN operator, specify the LEADING hint for the outer table and the INDEX hint for the inner table (the table with the spatial index), and specify the inner table last. For example:

```
SELECT /*+ LEADING(b) INDEX(a cola_spatial_idx) */ a.gid
      FROM cola_gry b, cola_markets a
      WHERE b.gid =1 AND
            SDO_NN(a.shape, b.shape, 'querytype=window sdo_num_res=1')='TRUE';
```

For detailed information about using optimizer hints, see *Oracle Database Performance Tuning Guide*.

Examples

The following example finds the two objects from the SHAPE column in the COLA_MARKETS table that are nearest to a specified point (10,7). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).)

```
SELECT /*+ INDEX(c cola_spatial_idx) */
      c.mkt_id, c.name FROM cola_markets c WHERE SDO_NN(c.shape,
      sdo_geometry(2001, NULL, sdo_point_type(10,7,NULL), NULL,
      NULL), 'sdo_num_res=2') = 'TRUE';
```

```
      MKT_ID NAME
-----
          2 cola_b
          4 cola_d
```

The following example uses the `sdo_batch_size` keyword to find the two objects (ROWNUM <=2), with a NAME value less than 'cola_d', from the SHAPE column in the COLA_MARKETS table that are nearest to a specified point (10,7). The value of 3 for `sdo_batch_size` represents a best guess at the number of nearest geometries that need to be evaluated before the WHERE clause condition is satisfied. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT /*+ INDEX(c cola_spatial_idx) */ c.mkt_id, c.name
      FROM cola_markets c
      WHERE SDO_NN(c.shape, sdo_geometry(2001, NULL,
      sdo_point_type(10,7,NULL), NULL, NULL),
      'sdo_batch_size=3') = 'TRUE'
      AND c.name < 'cola_d' AND ROWNUM <= 2;
```

```
      MKT_ID NAME
-----
          2 cola_b
          3 cola_c
```

See also the more complex SDO_NN examples in [Section D.3](#).

Related Topics

- [SDO_NN_DISTANCE](#)

SDO_NN_DISTANCE

Format

SDO_NN_DISTANCE(number);

Description

Returns the distance of an object returned by the [SDO_NN](#) operator. Valid only within a call to the [SDO_NN](#) operator.

Keywords and Parameters

Value	Description
number	Specifies a number that must be the same as the last parameter passed to the SDO_NN operator. Data type is NUMBER.

Returns

This operator returns the distance of an object returned by the [SDO_NN](#) operator. In determining how near two geometry objects are, the shortest possible distance between any two points on the surface of each object is used.

Usage Notes

SDO_NN_DISTANCE is an ancillary operator to the [SDO_NN](#) operator. It returns the distance between the specified geometry and a nearest neighbor object. This distance is passed as ancillary data to the [SDO_NN](#) operator. (For an explanation of how operators can use ancillary data, see the section on ancillary data in the chapter on domain indexes in *Oracle Data Cartridge Developer's Guide*.)

You can choose any arbitrary number for the `number` parameter. The only requirement is that it must match the last parameter in the call to the [SDO_NN](#) operator.

Use a bind variable to store and operate on the distance value.

Examples

The following example finds the two objects from the `SHAPE` column in the `COLA_MARKETS` table that are nearest to a specified point (10,7), and it finds the distance between each object and the point. (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).)

```
SELECT /*+ INDEX(c cola_spatial_idx) */
       c.mkt_id, c.name, SDO_NN_DISTANCE(1) dist
FROM   cola_markets c
WHERE  SDO_NN(c.shape, sdo_geometry(2001, NULL,
                                     sdo_point_type(10,7,NULL), NULL, NULL),
            'sdo_num_res=2', 1) = 'TRUE' ORDER BY dist;
```

MKT_ID	NAME	DIST
4	cola_d	.828427125
2	cola_b	2.23606798

Note the following about this example:

- 1 is used as the `number` parameter for `SDO_NN_DISTANCE`, and 1 is also specified as the last parameter to [SDO_NN](#) (after `'sdo_num_res=2'`).
- The column alias `dist` holds the distance between the object and the point. (For geodetic data, the distance unit is meters; for non-geodetic data, the distance unit is the unit associated with the data.)

Related Topics

- [SDO_NN](#)

SDO_ON

Format

```
SDO_ON(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the ON topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=ON'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_ON(geometry1,geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the ON topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the ON relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 4,6, 8,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) This example returns no rows because there are no line string geometries in the SHAPE column.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_ON(c.shape,
              SDO_GEOMETRY(2003, NULL, NULL,
                           SDO_ELEM_INFO_ARRAY(1,1003,3),
                           SDO_ORDINATE_ARRAY(4,6, 8,8))
              ) = 'TRUE';
```

```
no rows selected
```

SDO_OVERLAPBDYDISJOINT

Format

```
SDO_OVERLAPBDYDISJOINT(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the OVERLAPBDYDISJOINT topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=OVERLAPBDYDISJOINT'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_OVERLAPBDYDISJOINT(geometry1,geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the OVERLAPBDYDISJOINT topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the OVERLAPBDYDISJOINT relationship with a line string geometry (here, a horizontal line from 0,6 to 2,6). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, only `cola_a` has the OVERLAPBDYDISJOINT relationship with the line string geometry.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_OVERLAPBDYDISJOINT(c.shape,
                               SDO_GEOMETRY(2002, NULL, NULL,
                                             SDO_ELEM_INFO_ARRAY(1,2,1),
                                             SDO_ORDINATE_ARRAY(0,6, 2,6))
                               ) = 'TRUE';
```

```
MKT_ID NAME
-----
```

1 cola_a

SDO_OVERLAPBDYINTERSECT

Format

```
SDO_OVERLAPBDYINTERSECT(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the OVERLAPBDYINTERSECT topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=OVERLAPBDYINTERSECT'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_OVERLAPBDYINTERSECT(geometry1, geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the OVERLAPBDYINTERSECT topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the OVERLAPBDYINTERSECT relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 4,6,8,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, `cola_a`, `cola_b`, and `cola_d` have the OVERLAPBDYINTERSECT relationship with the query window geometry.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_OVERLAPBDYINTERSECT(c.shape,
                                SDO_GEOMETRY(2003, NULL, NULL,
                                              SDO_ELEM_INFO_ARRAY(1,1003,3),
                                              SDO_ORDINATE_ARRAY(4,6, 8,8))
                                ) = 'TRUE';

MKT_ID NAME
```

```
-----  
2 cola_b  
1 cola_a  
4 cola_d
```

SDO_OVERLAPS

Format

```
SDO_OVERLAPS(geometry1, geometry2);
```

Description

Checks if any geometries in a table overlap (that is, have the OVERLAPBDYDISJOINT or OVERLAPBDYINTERSECT topological relationship with) a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=OVERLAPBDYDISJOINT+OVERLAPBDYINTERSECT'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression SDO_OVERLAPS(geometry1,geometry2) = 'TRUE' evaluates to TRUE for object pairs that have the OVERLAPBDYDISJOINT or OVERLAPBDYINTERSECT topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that overlap a query window (here, a rectangle with lower-left, upper-right coordinates 4,6, 8,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, three of the geometries in the SHAPE column overlap the query window geometry.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_OVERLAPS(c.shape,
                    SDO_GEOMETRY(2003, NULL, NULL,
                                   SDO_ELEM_INFO_ARRAY(1,1003,3),
                                   SDO_ORDINATE_ARRAY(4,6, 8,8))
                    ) = 'TRUE';

MKT_ID NAME
```

2 cola_b
1 cola_a
4 cola_d

SDO_RELATE

Format

```
SDO_RELATE(geometry1, geometry2, param);
```

Description

Uses the spatial index to identify either the spatial objects that have a particular spatial interaction with a given object such as an area of interest, or pairs of spatial objects that have a particular spatial interaction.

This operator performs both primary and secondary filter operations.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.
param	Specifies the mask keyword, and optionally either or both of the <code>min_resolution</code> and <code>max_resolution</code> keywords. The data type for this parameter is VARCHAR2. The mask keyword specifies the topological relationship of interest. This is a required parameter. Valid mask keyword values are one or more of the following in the nine-intersection pattern: TOUCH, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, EQUAL, INSIDE, COVEREDBY, CONTAINS, COVERS, ANYINTERACT, ON. Multiple masks are combined with the logical Boolean operator OR, for example, 'mask=inside+touch'; however, see the Usage Notes for an alternative syntax using UNION ALL that may result in better performance. See Section 1.8 for an explanation of the nine-intersection relationship pattern. The <code>min_resolution</code> keyword includes only geometries for which at least one side of the geometry's MBR is equal to or greater than the specified value. For example, <code>min_resolution=10</code> includes only geometries for which the width or the height (or both) of the geometry's MBR is at least 10. (This keyword can be used to exclude geometries that are too small to be of interest.) The <code>max_resolution</code> keyword includes only geometries for which at least one side of the geometry's MBR is less than or equal to the specified value. For example, <code>max_resolution=10</code> includes only geometries for which the width or the height (or both) of the geometry's MBR is less than or equal to 10. (This keyword can be used to exclude geometries that are too large to be of interest.) For backward compatibility, any additional keywords for the param parameter that were supported before release 10.1 will still work; however, the use of those keywords is discouraged and is not supported for new uses of the operator.

Returns

The expression `SDO_RELATE(geometry1,geometry2, 'mask = <some_mask_val>') = 'TRUE'` evaluates to TRUE for object pairs that have the topological relationship specified by `<some_mask_val>`, and FALSE otherwise.

Usage Notes

The operator is disabled if the table does not have a spatial index or if the index has been built on more than two dimensions.

The operator must always be used in a WHERE clause, and the condition that includes the operator should be an expression of the form `SDO_RELATE(arg1, arg2, 'mask = <some_mask_val>') = 'TRUE'`.

`geometry2` can come from a table or be a transient `SDO_GEOMETRY` object (such as a bind variable or `SDO_GEOMETRY` constructor).

- If the `geometry2` column is not spatially indexed, the operator indexes the query window in memory and performance is very good.
- If two or more geometries from `geometry2` are passed to the operator, the `ORDERED` optimizer hint must be specified, and the table in `geometry2` must be specified first in the FROM clause.

If `geometry1` and `geometry2` are based on different coordinate systems, `geometry2` is temporarily transformed to the coordinate system of `geometry1` for the operation to be performed, as described in [Section 6.9.1](#).

Unlike with the [SDO_GEOM.RELATE](#) function, `DISJOINT` and `DETERMINE` masks are not allowed in the relationship mask with the `SDO_RELATE` operator. This is because `SDO_RELATE` uses the spatial index to find candidates that may interact, and the information to satisfy `DISJOINT` or `DETERMINE` is not present in the index.

Although multiple masks can be combined using the logical Boolean operator `OR`, for example, `'mask=touch+coveredby'`, better performance may result if the spatial query specifies each mask individually and uses the `UNION ALL` syntax to combine the results. This is due to internal optimizations that Spatial can apply under certain conditions when masks are specified singly rather than grouped within the same `SDO_RELATE` operator call. (There are two exceptions, `inside+coveredby` and `contains+covers`, where the combination performs better than the `UNION ALL` alternative.) For example, consider the following query using the logical Boolean operator `OR` to group multiple masks:

```
SELECT a.gid
  FROM polygons a, query_polys B
 WHERE B.gid = 1
 AND SDO_RELATE(A.Geometry, B.Geometry,
                'mask=touch+coveredby') = 'TRUE';
```

The preceding query may result in better performance if it is expressed as follows, using `UNION ALL` to combine results of multiple `SDO_RELATE` operator calls, each with a single mask:

```
SELECT a.gid
  FROM polygons a, query_polys B
 WHERE B.gid = 1
 AND SDO_RELATE(A.Geometry, B.Geometry,
                'mask=touch') = 'TRUE'

UNION ALL

SELECT a.gid
  FROM polygons a, query_polys B
```

```
WHERE B.gid = 1
AND SDO_RELATE(A.Geometry, B.Geometry,
               'mask=coveredby') = 'TRUE';
```

The following considerations apply to relationships between lines and a multipoint geometry (points in a point cluster). Assume the example of a line and a multipoint geometry (for example, SDO_GTYPE = 2005) consisting of three points.

- If none of the points has any interaction with the line, the relationship between the line and the point cluster is DISJOINT.
- If one of the points is on the interior of the line and the other two points are disjoint, the relationship between the line and the point cluster is OVERLAPBDYDISJOINT.
- If one of the points is on the boundary of the line (that is, if it is on the start point or end point of the line) and the other two points are disjoint, the relationship between the line and the point cluster is TOUCH.
- If one of the points is on the boundary of the line (that is, if it is on the start point or end point of the line), another point is on the interior of the line, and the third point is disjoint, the relationship between the line and the point cluster is OVERLAPBDYDISJOINT (not OVERLAPBDYINTERSECT).

Examples

The following examples are similar to those for the [SDO_FILTER](#) operator; however, they identify a specific type of interaction (using the `mask` keyword), and they determine with certainty (not mere likelihood) if the spatial interaction occurs.

The following example selects the geometries that have any interaction with a query window (here, a rectangle with lower-left, upper-right coordinates 4,6, 8,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#).)

```
SELECT c.mkt_id, c.name
FROM cola_markets c
WHERE SDO_RELATE(c.shape,
                 SDO_GEOMETRY(2003, NULL, NULL,
                               SDO_ELEM_INFO_ARRAY(1,1003,3),
                               SDO_ORDINATE_ARRAY(4,6, 8,8)),
                 'mask=anyinteract') = 'TRUE';
```

```
MKT_ID NAME
-----
      2 cola_b
      1 cola_a
      4 cola_d
```

The following example is the same as the preceding example, except that it includes only geometries where at least one side of the geometry's MBR is equal to or greater than 4.1. In this case, only `cola_a` and `cola_b` are returned, because their MBRs have at least one side with a length greater than or equal to 4.1. The circle `cola_d` is excluded, because its MBR is a square whose sides have a length of 4.

```
SELECT c.mkt_id, c.name
FROM cola_markets c
WHERE SDO_RELATE(c.shape,
                 SDO_GEOMETRY(2003, NULL, NULL,
                               SDO_ELEM_INFO_ARRAY(1,1003,3),
                               SDO_ORDINATE_ARRAY(4,6, 8,8)),
                 'mask=anyinteract') = 'TRUE';
```

```
'mask=anyinteract min_resolution=4.1') = 'TRUE';
```

```
MKT_ID NAME
```

```
-----
      2 cola_b
      1 cola_a
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column objects have any spatial interaction with the GEOMETRY column object in the QUERY_POLYS table that has a GID value of 1.

```
SELECT A.gid
   FROM Polygons A, query_polys B
  WHERE B.gid = 1
     AND SDO_RELATE(A.Geometry, B.Geometry,
                   'mask=ANYINTERACT') = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where a GEOMETRY column object has any spatial interaction with the geometry stored in the aGeom variable.

```
SELECT A.Gid
   FROM Polygons A
  WHERE SDO_RELATE(A.Geometry, :aGeom, 'mask=ANYINTERACT') = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where a GEOMETRY column object has any spatial interaction with the specified rectangle having the lower-left coordinates (x1,y1) and the upper-right coordinates (x2, y2).

```
SELECT A.Gid
   FROM Polygons A
  WHERE SDO_RELATE(A.Geometry, sdo_geometry(2003,NULL,NULL,
                                           sdo_elem_info_array(1,1003,3),
                                           sdo_ordinate_array(x1,y1,x2,y2)),
                   'mask=ANYINTERACT') = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column object has any spatial interaction with any GEOMETRY column object in the QUERY_POLYS table. In this example, the ORDERED optimizer hint is used and QUERY_POLYS (geometry2) table is specified first in the FROM clause, because multiple geometries from geometry2 are involved (see the Usage Notes).

```
SELECT /*+ ORDERED */
      A.gid
   FROM query_polys B, polygons A
  WHERE SDO_RELATE(A.Geometry, B.Geometry, 'mask=ANYINTERACT') = 'TRUE';
```

Related Topics

- [SDO_FILTER](#)
- [SDO_JOIN](#)
- [SDO_WITHIN_DISTANCE](#)
- [SDO_GEOM.RELATE](#) function

SDO_TOUCH

Format

```
SDO_TOUCH(geometry1, geometry2);
```

Description

Checks if any geometries in a table have the TOUCH topological relationship with a specified geometry. Equivalent to specifying the [SDO_RELATE](#) operator with 'mask=TOUCH'.

See the section on the [SDO_RELATE](#) operator in this chapter for information about the operations performed by this operator and for usage requirements.

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column must be spatially indexed. Data type is SDO_GEOMETRY.
geometry2	Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or SDO_GEOMETRY constructor.) Data type is SDO_GEOMETRY.

Returns

The expression `SDO_TOUCH(geometry1,geometry2) = 'TRUE'` evaluates to TRUE for object pairs that have the TOUCH topological relationship, and FALSE otherwise.

Usage Notes

See the Usage Notes for the [SDO_RELATE](#) operator in this chapter.

For an explanation of the topological relationships and the nine-intersection model used by Spatial, see [Section 1.8](#).

Examples

The following example finds geometries that have the TOUCH relationship with a query window (here, a rectangle with lower-left, upper-right coordinates 1,1, 5,7). (The example uses the definitions and data in [Section 2.1](#) and illustrated in [Figure 2-1](#).) In this example, only cola_b has the TOUCH relationship with the query window geometry.

```
SELECT c.mkt_id, c.name
   FROM cola_markets c
  WHERE SDO_TOUCH(c.shape,
                 SDO_GEOMETRY(2003, NULL, NULL,
                               SDO_ELEM_INFO_ARRAY(1,1003,3),
                               SDO_ORDINATE_ARRAY(1,1, 5,7))
                 ) = 'TRUE';
   FROM cola_markets c

      MKT_ID NAME
-----
```

2 cola_b

SDO_WITHIN_DISTANCE

Format

```
SDO_WITHIN_DISTANCE(geometry1, aGeom, params);
```

Description

Uses the spatial index to identify the set of spatial objects that are within some specified distance of a given object (such as an area of interest or point of interest).

Keywords and Parameters

Value	Description
geometry1	Specifies a geometry column in a table. The column has the set of geometry objects that will be operated on to determine if they are within the specified distance of the given object (aGeom). The column must be spatially indexed. Data type is SDO_GEOMETRY.
aGeom	Specifies the object to be checked for distance against the geometry objects in geometry1. Specify either a geometry from a table (using a bind variable) or a transient instance of a geometry (using the SDO_GEOMETRY constructor). Data type is SDO_GEOMETRY.
params	A quoted string containing one or more keywords (with values) that determine the behavior of the operator. The remaining items (distance, max_resolution, min_resolution, querytype, and unit) are potential keywords for the params parameter. Data type is VARCHAR2.
distance	Specifies the distance value. If a coordinate system is associated with the geometry, the distance unit is assumed to be the unit associated with the coordinate system. This is a required keyword. Data type is NUMBER.
max_resolution	Includes only geometries for which at least one side of the geometry's MBR is less than or equal to the specified value. For example, max_resolution=10 includes only geometries for which the width or the height (or both) of the geometry's MBR is less than or equal to 10. (This keyword can be used to exclude geometries that are too large to be of interest.)
min_resolution	Includes only geometries for which at least one side of the geometry's MBR is equal to or greater than the specified value. For example, min_resolution=10 includes only geometries for which the width or the height (or both) of the geometry's MBR is at least 10. (This keyword can be used to exclude geometries that are too small to be of interest.)
querytype	Set 'querytype=FILTER' to perform only a primary filter operation. If querytype is not specified, both primary and secondary filter operations are performed (default). Data type is VARCHAR2.
unit	Specifies the unit of measurement: a quoted string with unit= and an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table (for example, 'unit=KM'). See Section 2.8 for more information about unit of measurement specification. Data type is NUMBER. Default = unit of measurement associated with the data. For geodetic data, the default is meters.

Returns

The expression `SDO_WITHIN_DISTANCE(arg1, arg2, arg3) = 'TRUE'` evaluates to `TRUE` for object pairs that are within the specified distance, and `FALSE` otherwise.

Usage Notes

The distance between two extended objects (nonpoint objects such as lines and polygons) is defined as the minimum distance between these two objects. The distance between two adjacent polygons is zero.

If this operator is used with geodetic data, the data must be indexed with an R-tree spatial index. If this operator is used with geodetic data and if the R-tree spatial index is created with `'geodetic=false'` specified, you cannot use the `unit` parameter.

The operator is disabled if the table does not have a spatial index or if the index has been built on more than two dimensions.

The operator must always be used in a `WHERE` clause and the condition that includes the operator should be an expression of the form:

```
SDO_WITHIN_DISTANCE(arg1, arg2, 'distance = <some_dist_val>') = 'TRUE'
```

The geometry column must have a spatial index built on it. If the data is geodetic, the spatial index must be an R-tree index.

`SDO_WITHIN_DISTANCE` is not supported for spatial joins. See [Section 4.2.1.3](#) for a discussion on how to perform a spatial join within-distance operation.

Examples

The following example selects the geometries that are within a distance of 10 from a query window (here, a rectangle with lower-left, upper-right coordinates 4,6, 8,8). (The example uses the definitions and data described in [Section 2.1](#) and illustrated in [Figure 2-1](#). In this case, all geometries shown in that figure are returned.)

```
SELECT c.name FROM cola_markets c WHERE SDO_WITHIN_DISTANCE(c.shape,
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
  SDO_ORDINATE_ARRAY(4,6, 8,8)),
  'distance=10') = 'TRUE';
```

```
NAME
-----
cola_b
cola_a
cola_c
cola_d
```

The following example is the same as the preceding example, except that it includes only geometries where at least one side of the geometry's MBR is equal to or greater than 4.1. In this case, only `cola_a` and `cola_b` are returned, because their MBRs have at least one side with a length greater than or equal to 4.1. The trapezoid `cola_c` is excluded, because its MBR has sides with lengths of 3 and 2; and the circle `cola_d` is excluded, because its MBR is a square whose sides have a length of 4.

```
SELECT c.name FROM cola_markets c WHERE SDO_WITHIN_DISTANCE(c.shape,
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
  SDO_ORDINATE_ARRAY(4,6, 8,8)),
  'distance=10 min_resolution=4.1') = 'TRUE';
```

```
NAME
-----
```

```
cola_b
cola_a
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column object is within 10 distance units of the geometry stored in the aGeom variable.

```
SELECT A.GID
FROM POLYGONS A
WHERE
  SDO_WITHIN_DISTANCE(A.Geometry, :aGeom, 'distance = 10') = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where the GEOMETRY column object is within 10 distance units of the specified rectangle having the lower-left coordinates (x1,y1) and the upper-right coordinates (x2, y2).

```
SELECT A.GID
FROM POLYGONS A
WHERE
  SDO_WITHIN_DISTANCE(A.Geometry, sdo_geometry(2003,NULL,NULL,
    sdo_elem_info_array(1,1003,3),
    sdo_ordinate_array(x1,y1,x2,y2)),
    'distance = 10') = 'TRUE';
```

The following example selects the GID values from the POLYGONS table where the GID value in the QUERY_POINTS table is 1 and a POLYGONS.GEOMETRY object is within 10 distance units of the QUERY_POINTS.GEOMETRY object.

```
SELECT A.GID
FROM POLYGONS A, Query_Points B
WHERE B.GID = 1 AND
  SDO_WITHIN_DISTANCE(A.Geometry, B.Geometry, 'distance = 10') = 'TRUE';
```

See also the more complex SDO_WITHIN_DISTANCE examples in [Section D.2](#).

Related Topics

- [SDO_FILTER](#)
- [SDO_RELATE](#)

Spatial Aggregate Functions

This chapter contains reference and usage information for the spatial aggregate functions, which are listed in [Table 12-1](#).

Table 12-1 *Spatial Aggregate Functions*

Method	Description
SDO_AGGR_CENTROID	Returns a geometry object that is the centroid ("center of gravity") of the specified geometry objects.
SDO_AGGR_CONCAT_LINES	Returns a geometry that concatenates the specified line or multiline geometries.
SDO_AGGR_CONVEXHULL	Returns a geometry object that is the convex hull of the specified geometry objects.
SDO_AGGR_LRS_CONCAT	Returns an LRS geometry object that concatenates specified LRS geometry objects.
SDO_AGGR_MBR	Returns the minimum bounding rectangle of the specified geometries.
SDO_AGGR_UNION	Returns a geometry object that is the topological union (<i>OR</i> operation) of the specified geometry objects.

See the usage information about spatial aggregate functions in [Section 1.10](#).

Most of these aggregate functions accept a parameter of type `SDOAGGRTYPE`, which is described in [Section 1.10.1](#).

SDO_AGGR_CENTROID

Format

```
SDO_AGGR_CENTROID(
  AggregateGeometry SDOAGGRTYPE
) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry object that is the centroid ("center of gravity") of the specified geometry objects.

Parameters

AggregateGeometry

An object of type SDOAGGRTYPE (see [Section 1.10.1](#)) that specifies the geometry column and dimensional array.

Usage Notes

The behavior of the function depends on whether the geometry objects are all polygons, all points, or a mixture of polygons and points:

- If the geometry objects are all polygons, the centroid of all the objects is returned.
- If the geometry objects are all points, the centroid of all the objects is returned.
- If the geometry objects are a mixture of polygons and points (specifically, if they include at least one polygon and at least one point), any points are ignored, and the centroid of all the polygons is returned.

The result is weighted by the area of each polygon in the geometry objects. If the geometry objects are a mixture of polygons and points, the points are not used in the calculation of the centroid. If the geometry objects are all points, the points have equal weight.

See also the information about the [SDO_GEOM.SDO_CENTROID](#) function in [Chapter 15](#).

Examples

The following example returns the centroid of the geometry objects in the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT SDO_AGGR_CENTROID(SDOAGGRTYPE(shape, 0.005))
  FROM cola_markets;

SDO_AGGR_CENTROID(SDOAGGRTYPE(SHAPE,0.005)) (SDO_GTYPE, SDO_SRID, SDO_POINT
-----
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(5.21295938, 5.00744233, NULL), NULL, NUL
L)
```

SDO_AGGR_CONCAT_LINES

Format

```
SDO_AGGR_CONCAT_LINES(
    geom SDO_GEOMETRY
) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry that concatenates the specified line or multiline geometries.

Parameters

geom
Geometry objects.

Usage Notes

Each input geometry must be a two-dimensional line or multiline geometry (that is, the SDO_GTYPE value must be 2002 or 2006). This function is not supported for LRS geometries. To perform an aggregate concatenation of LRS geometric segments, use the [SDO_AGGR_LRS_CONCAT](#) spatial aggregate function.

The input geometries must be line strings whose vertices are connected by straight line segments. Circular arcs and compound line strings are not supported.

If any input geometry is a multiline geometry, the elements of the geometry must be disjoint. If they are not disjoint, this function may return incorrect results.

The topological relationship between the geometries in each pair of geometries to be concatenated must be DISJOINT or TOUCH; and if the relationship is TOUCH, the geometries must intersect only at two end points.

You can use the [SDO_UTIL.CONCAT_LINES](#) function (described in [Chapter 20](#)) to concatenate two line or multiline geometries.

An exception is raised if any input geometries are not line or multiline geometries, or if not all input geometries are based on the same coordinate system.

Examples

The following example inserts two line geometries in the COLA_MARKETS table, and then returns the aggregate concatenation of these geometries. (The example uses the data definitions from [Section 2.1](#).)

```
-- First, insert two line geometries.
INSERT INTO cola_markets VALUES(1001, 'line_1', SDO_GEOMETRY(2002, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1), SDO_ORDINATE_ARRAY(1,1, 5,1)));
INSERT INTO cola_markets VALUES(1002, 'line_2', SDO_GEOMETRY(2002, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1), SDO_ORDINATE_ARRAY(5,1, 8,1)));
-- Perform aggregate concatenation of all line geometries in layer.
SELECT SDO_AGGR_CONCAT_LINES(c.shape) FROM cola_markets c
    WHERE c.mkt_id > 1000;

SDO_AGGR_CONCAT_LINES(C.SHAPE)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM
-----
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
```

1, 1, 5, 1, 8, 1))

SDO_AGGR_CONVEXHULL

Format

```
SDO_AGGR_CONVEXHULL(
    AggregateGeometry SDOAGGRTYPE
) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry object that is the convex hull of the specified geometry objects.

Parameters

AggregateGeometry

An object of type SDOAGGRTYPE (see [Section 1.10.1](#)) that specifies the geometry column and dimensional array.

Usage Notes

See also the information about the [SDO_GEOM.SDO_CONVEXHULL](#) function in [Chapter 15](#).

Examples

The following example returns the convex hull of the geometry objects in the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(shape, 0.005))
       FROM cola_markets;

SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(SHAPE,0.005))(SDO_GTYPE, SDO_SRID, SDO_POI
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(8, 1, 10, 7, 10, 11, 8, 11, 6, 11, 1, 7, 1, 1, 8, 1))
```

SDO_AGGR_LRS_CONCAT

Format

```
SDO_AGGR_LRS_CONCAT(
  AggregateGeometry SDOAGGRTYPE
) RETURN SDO_GEOMETRY;
```

Description

Returns an LRS geometry that concatenates specified LRS geometries.

Parameters

AggregateGeometry

An object of type SDOAGGRTYPE (see [Section 1.10.1](#)) that specifies the geometry column and dimensional array.

Usage Notes

This function performs an aggregate concatenation of any number of LRS geometries. If you want to control the order in which the geometries are concatenated, you must use a subquery with the NO_MERGE optimizer hint and the ORDER BY clause. (See the examples.)

The direction of the resulting segment is the same as the direction of the first geometry in the concatenation.

A 3D format of this function (SDO_AGGR_LRS_CONCAT_3D) is available. For information about 3D formats of LRS functions, see [Section 7.4](#).)

For information about the Spatial linear referencing system, see [Chapter 7](#).

Examples

The following example adds an LRS geometry to the LRS_ROUTES table, and then performs two queries that concatenate the LRS geometries in the table. The first query does not control the order of concatenation, and the second query controls the order of concatenation. Notice the difference in direction of the two segments: the segment resulting from the second query has decreasing measure values because the first segment in the concatenation (Route0) has decreasing measure values. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- Add a segment with route_id less than 1 (here, zero).
INSERT INTO lrs_routes VALUES(
  0,
  'Route0',
  SDO_GEOMETRY(
    3302, -- Line string; 3 dimensions (X,Y,M); 3rd is measure dimension.
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1), -- One line string, straight segments
    SDO_ORDINATE_ARRAY(
      5,14,5, -- Starting point - 5 is measure from start.
      10,14,0) -- Ending point - 0 measure (decreasing measure)
  )
);
```

1 row created.

-- Concatenate all routes (no ordering specified).

```
SELECT SDO_AGGR_LRS_CONCAT(SDOAGGRTYPE(route_geometry, 0.005))
      FROM lrs_routes;
```

```
SDO_AGGR_LRS_CONCAT(SDOAGGRTYPE(ROUTE_GEOMETRY,0.005))(SDO_GTYPE, SDO_SRID
```

```
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27, 10, 14,
32))
```

-- Aggregate concatenation using subquery for ordering.

```
SELECT
SDO_AGGR_LRS_CONCAT(SDOAGGRTYPE(route_geometry, 0.005))
FROM (
      SELECT /*+ NO_MERGE */ route_geometry
      FROM lrs_routes
      ORDER BY route_id);
```

```
SDO_AGGR_LRS_CONCAT(SDOAGGRTYPE(ROUTE_GEOMETRY,0.005))(SDO_GTYPE, SDO_SRID
```

```
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 32, 2, 4, 30, 8, 4, 24, 12, 4, 20, 12, 10, 14, 8, 10, 10, 5, 14, 5, 10, 14
, 0))
```

SDO_AGGR_MBR

Format

```
SDO_AGGR_MBR(
  geom SDO_GEOMETRY
) RETURN SDO_GEOMETRY;
```

Description

Returns the minimum bounding rectangle (MBR) of the specified geometries, that is, a single rectangle that minimally encloses the geometries.

Parameters

geom
Geometry objects.

Usage Notes

All input geometries must have 4-digit SDO_GTYPE values (explained in [Section 2.2.1](#)).

This function does not return an MBR geometry if a proper MBR cannot be constructed. Specifically:

- If the input geometries are all null, the function returns a null geometry.
- If all data in the input geometries is on a single point, the function returns the point.
- If all data in the input geometries consists of points on a straight line, the function returns a two-point line.

The [SDO_TUNE.EXTENT_OF](#) function, documented in [Chapter 19](#), also returns the MBR of geometries. The [SDO_TUNE.EXTENT_OF](#) function has better performance than the SDO_AGGR_MBR function if the data is non-geodetic and if a spatial index is defined on the geometry column; however, the [SDO_TUNE.EXTENT_OF](#) function is limited to two-dimensional geometries, whereas the SDO_AGGR_MBR function is not. In addition, the [SDO_TUNE.EXTENT_OF](#) function computes the extent for all geometries in a table; by contrast, the SDO_AGGR_MBR function can operate on subsets of rows.

Examples

The following example returns the minimum bounding rectangle of the geometry objects in the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT SDO_AGGR_MBR(shape) FROM cola_markets;

SDO_AGGR_MBR(C.SHAPE) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SD
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(1, 1, 10, 11))
```

SDO_AGGR_UNION

Format

```
SDO_AGGR_UNION(
    AggregateGeometry SDOAGGRTYPE
) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry object that is the topological union (OR operation) of the specified geometry objects.

Parameters

AggregateGeometry

An object of type SDOAGGRTYPE (see [Section 1.10.1](#)) that specifies the geometry column and dimensional array.

Usage Notes

Do not use SDO_AGGR_UNION to merge line string or multiline string geometries; instead, use the [SDO_AGGR_CONCAT_LINES](#) spatial aggregate function.

See also the information about the [SDO_GEOM.SDO_UNION](#) function in [Chapter 15](#).

Examples

The following example returns the union of the first three geometry objects in the COLA_MARKETS table (that is, all except cola_d). (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT SDO_AGGR_UNION(
    SDOAGGRTYPE(c.shape, 0.005))
FROM cola_markets c
WHERE c.name < 'cola_d';

SDO_AGGR_UNION(SDOAGGRTYPE(C.SHAPE,0.005))(SDO_GTYPE, SDO_SRID, SDO_POINT(
-----
SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 2, 11, 1003, 1), SDO
_ORDINATE_ARRAY(8, 11, 6, 9, 8, 7, 10, 9, 8, 11, 1, 7, 1, 1, 5, 1, 8, 1, 8, 6, 5
, 7, 1, 7))
```

See also the more complex SDO_AGGR_UNION example in [Section D.4](#).

SDO_CS Package (Coordinate System Transformation)

The MDSYS.SDO_CS package contains subprograms for working with coordinate systems. You can perform explicit coordinate transformations on a single geometry or an entire layer of geometries (that is, all geometries in a specified column in a table).

To use the subprograms in this chapter, you must understand the conceptual information about coordinate systems in [Section 1.5.4](#) and [Chapter 6](#).

[Table 13–1](#) lists the coordinate system transformation subprograms.

Table 13–1 Subprograms for Coordinate System Transformation

Subprogram	Description
SDO_CS.ADD_PREFERENCE_FOR_OP	Adds a preference for an operation between a source coordinate system and a target coordinate system.
SDO_CS.CONVERT_NADCON_TO_XML	Converts a NADCON (North American Datum Conversion) grid in ASCII format to an Oracle Spatial XML representation.
SDO_CS.CONVERT_NTV2_TO_XML	Converts an NTv2 (National Transformation Version 2) grid in ASCII format to an Oracle Spatial XML representation.
SDO_CS.CONVERT_XML_TO_NADCON	Converts an Oracle Spatial XML representation of a NADCON (North American Datum Conversion) grid to NADCON ASCII format.
SDO_CS.CONVERT_XML_TO_NTV2	Converts an Oracle Spatial XML representation of an NTv2 (National Transformation Version 2) grid to NTv2 ASCII format.
SDO_CS.CREATE_CONCATENATED_OP	Creates a concatenated operation.
SDO_CS.CREATE_OBVIOUS_EPSG_RULES	Creates a basic set of EPSG rules to be applied in certain transformations.
SDO_CS.CREATE_PREF_CONCATENATED_OP	Creates a concatenated operation, associating it with a transformation plan and making it preferred either systemwide or for a specified use case.
SDO_CS.DELETE_ALL_EPSG_RULES	Deletes the basic set of EPSG rules to be applied in certain transformations.
SDO_CS.DELETE_OP	Deletes a concatenated operation.

Table 13–1 (Cont.) Subprograms for Coordinate System Transformation

Subprogram	Description
SDO_CS.DETERMINE_CHAIN	Returns the query chain, based on the system rule set, to be used in transformations from one coordinate reference system to another coordinate reference system.
SDO_CS.DETERMINE_DEFAULT_CHAIN	Returns the default chain of SRID values in transformations from one coordinate reference system to another coordinate reference system.
SDO_CS.FIND_GEOG_CRS	Returns the SRID values of geodetic (geographic) coordinate reference systems that have the same well-known text (WKT) numeric values as the coordinate reference system with the specified reference SRID value.
SDO_CS.FIND_PROJ_CRS	Returns the SRID values of projected coordinate reference systems that have the same well-known text (WKT) numeric values as the coordinate reference system with the specified reference SRID value.
SDO_CS.FROM_OGC_SIMPLEFEATURE_SRS	Converts a well-known text string from the Open Geospatial Consortium simple feature format without the TOWGS84 keyword to the format that includes the TOWGS84 keyword.
SDO_CS.MAP_EPSG_SRID_TO_ORACLE	Returns the ORacle Spatial SRID values corresponding to the specified EPSG SRID value.
SDO_CS.MAP_ORACLE_SRID_TO_EPSG	Returns the EPSG SRID value corresponding to the specified Oracle Spatial SRID value.
SDO_CS.REVOKE_PREFERENCE_FOR_OP	Revokes a preference for an operation between a source coordinate system and a target coordinate system.
SDO_CS.TO_OGC_SIMPLEFEATURE_SRS	Converts a well-known text string from the Open Geospatial Consortium simple feature format that includes the TOWGS84 keyword to the format without the TOWGS84 keyword.
SDO_CS.TRANSFORM	Transforms a geometry representation using a coordinate system (specified by SRID or name).
SDO_CS.TRANSFORM_LAYER	Transforms an entire layer of geometries (that is, all geometries in a specified column in a table).
SDO_CS.UPDATE_WKTS_FOR_ALL_EPSG_CRS	Updates the well-known text (WKT) description for all EPSG coordinate reference systems.
SDO_CS.UPDATE_WKTS_FOR_EPSG_CRS	Updates the well-known text (WKT) description for the EPSG coordinate reference system associated with a specified SRID.
SDO_CS.UPDATE_WKTS_FOR_EPSG_DATUM	Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified datum.
SDO_CS.UPDATE_WKTS_FOR_EPSG_ELLIPS	Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified ellipsoid.
SDO_CS.UPDATE_WKTS_FOR_EPSG_OP	Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified coordinate transformation operation.

Table 13–1 (Cont.) Subprograms for Coordinate System Transformation

Subprogram	Description
SDO_CS.UPDATE_WKTS_FOR_EPSG_PARAM	Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified coordinate transformation operation and parameter for transformation operations.
SDO_CS.UPDATE_WKTS_FOR_EPSG_PM	Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified prime meridian.
SDO_CS.VALIDATE_WKT	Validates the well-known text (WKT) description associated with a specified SRID.
SDO_CS.VIEWPORT_TRANSFORM (deprecated)	Transforms an optimized rectangle into a valid polygon for use with Spatial operators and functions.

The rest of this chapter provides reference information on the subprograms, listed in alphabetical order.

SDO_CS.ADD_PREFERENCE_FOR_OP

Format

```
SDO_CS.ADD_PREFERENCE_FOR_OP(  
    op_id      IN NUMBER,  
    source_crs IN NUMBER DEFAULT NULL,  
    target_crs IN NUMBER DEFAULT NULL,  
    use_case   IN VARCHAR2 DEFAULT NULL);
```

Description

Adds a preference for an operation between a source coordinate system and a target coordinate system.

Parameters

op_id

ID number of the operation. Must be a value in the COORD_OP_ID column of the SDO_COORD_OPS table (described in [Section 6.6.8](#)).

source_crs

The SRID of the source coordinate reference system. Must be null or a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

target_crs

The SRID of the target coordinate reference system. Must be null or a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

use_case

Name of the use case to be associated with this preference. Must be null or a value from the USE_CASE column of the SDO_PREFERRED_OPS_USER table (described in [Section 6.6.25](#)).

Usage Notes

If `use_case` is null, the transformation plan associated with the operation is a systemwide preference, and a row is added (or two rows are added if a reverse operation exists) to the SDO_PREFERRED_OPS_SYSTEM table (described in [Section 6.6.24](#)). If `use_case` is not null, the transformation plan associated with the operation is a preference associated with the specified use case, and a row is added (or two rows are added if a reverse operation exists) to the SDO_PREFERRED_OPS_USER table (described in [Section 6.6.25](#)).

To create a concatenated operation and make it preferred either systemwide or for a specified use case, you can use the [SDO_CS.CREATE_PREF_CONCATENATED_OP](#) convenience procedure.

To revoke a preference for an operation between a source coordinate system and a target coordinate system, use the [SDO_CS.REVOKE_PREFERENCE_FOR_OP](#) procedure.

Examples

The following example adds a preference for operation 19977 to be used in transformations from SRID 4301 to SRID 4326 when use case use_case_B is specified for the transformation.

```
EXECUTE SDO_CS.ADD_PREFERENCE_FOR_OP(19977, 4301, 4326, 'use_case_B');
```

SDO_CS.CONVERT_NADCON_TO_XML

Format

```
SDO_CS.CONVERT_NADCON_TO_XML(  
    laa_clob IN CLOB,  
    loa_clob IN CLOB,  
    xml_grid OUT XMLTYPE );
```

Description

Converts a NADCON (North American Datum Conversion) grid in ASCII format to an Oracle Spatial XML representation.

Parameters

laa_clob

Latitude values of the NADCON grid in a CLOB object.

loa_clob

Longitude values of the NADCON grid in a CLOB object.

xml_grid

Output XML document containing the Oracle Spatial XML representation of the NADCON grid.

Usage Notes

To convert an Oracle Spatial XML representation to a NADCON grid, use the [SDO_CS.CONVERT_XML_TO_NADCON](#) procedure.

Examples

The following example converts a NADCON grid in ASCII format to an Oracle Spatial XML representation, converts the resulting XML representation back to a NADCON ASCII representation, and displays the resulting ASCII representation. (Only part of the output is shown.)

```
set lines 32000  
set long 2000000000  
  
DECLARE  
    laa      CLOB;  
    loa      CLOB;  
    xml      XMLTYPE;  
    laa_file BFILE;  
    loa_file BFILE;  
BEGIN  
    laa_file := BFILENAME('MY_WORK_DIR', 'samplenadcon.laa');  
    loa_file := BFILENAME('MY_WORK_DIR', 'samplenadcon.loa');  
    DBMS_LOB.OPEN(laa_file, DBMS_LOB.LOB_READONLY);  
    DBMS_LOB.OPEN(loa_file, DBMS_LOB.LOB_READONLY);  
  
    DBMS_LOB.CREATETEMPORARY(laa, TRUE, DBMS_LOB.SESSION);  
    DBMS_LOB.CREATETEMPORARY(loa, TRUE, DBMS_LOB.SESSION);  
    DBMS_LOB.OPEN(laa, DBMS_LOB.LOB_READWRITE);
```

```

DBMS_LOB.OPEN(lob, DBMS_LOB.LOB_READWRITE);
DBMS_LOB.LOADFROMFILE(lob, lob_file, DBMS_LOB.LOBMAXSIZE);
DBMS_LOB.LOADFROMFILE(lob, lob_file, DBMS_LOB.LOBMAXSIZE);
DBMS_LOB.CLOSE(lob);
DBMS_LOB.CLOSE(lob);
DBMS_LOB.CLOSE(lob_file);
DBMS_LOB.CLOSE(lob_file);

SDO_CS.convert_NADCON_to_XML(lob, lob, xml);
SDO_CS.convert_XML_to_NADCON(xml, lob, lob);
DBMS_OUTPUT.PUT_LINE(SUBSTR(lob, 1, 32000));
DBMS_OUTPUT.PUT_LINE(SUBSTR(lob, 1, 32000));
END;
/
NADCON EXTRACTED REGION                                NADGRD
33 49 1 -107.00000      .25000    25.00000      .25000      .00000
      .006731      .006444      .006208      .006036      .005935      .005904
      .005932      .006002      .006092      .006174      .006218      .006198
      .006087      .005867      .005522      .005045      .004432      .003688
      .002818      .001836      .000759      -.000385      -.001559      -.002704
. . .
NADCON EXTRACTED REGION                                NADGRD
33 49 1 -107.00000      .25000    25.00000      .25000      .00000
      .008509      .007147      .005756      .004331      .002879      .001410
      -.000060      -.001507      -.002904      -.004222      -.005431      -.006498
      -.007395      -.008095      -.008579      -.008832      -.008848      -.008632
      -.008200      -.007577      -.006800      -.005911      -.004957      -.003974
. . .

```

SDO_CS.CONVERT_NTV2_TO_XML

Format

```
SDO_CS.CONVERT_NTV2_TO_XML(  
    ntv2_clob IN CLOB,  
    xml_grid OUT XMLTYPE );
```

Description

Converts an NTv2 (National Transformation Version 2) grid in ASCII format to an Oracle Spatial XML representation.

Parameters

ntv2_clob

NTv2 grid values in a CLOB object.

xml_grid

Output XML document containing the Oracle Spatial XML representation of the NTv2 grid.

Usage Notes

To convert an Oracle Spatial XML representation to an NTv2 grid, use the [SDO_CS.CONVERT_XML_TO_NTV2](#) procedure.

Examples

The following example converts an NTv2 grid in ASCII format to an Oracle Spatial XML representation, converts the resulting XML representation back to an NTv2 ASCII representation, and displays the resulting ASCII representation. (Only part of the output is shown.)

```
set lines 32000  
set long 2000000000  
  
DECLARE  
    ntv2          CLOB;  
    xml           XMLTYPE;  
    ntv2_file     BFILE;  
BEGIN  
    ntv2_file := BFILENAME('MY_WORK_DIR', 'samplentv2.gsa');  
    DBMS_LOB.OPEN(ntv2_file, DBMS_LOB.LOB_READONLY);  
  
    DBMS_LOB.CREATETEMPORARY(ntv2, TRUE, DBMS_LOB.SESSION);  
    DBMS_LOB.OPEN(ntv2, DBMS_LOB.LOB_READWRITE);  
    DBMS_LOB.LOADFROMFILE(ntv2, ntv2_file, DBMS_LOB.LOBMAXSIZE);  
    DBMS_LOB.CLOSE(ntv2);  
    DBMS_LOB.CLOSE(ntv2_file);  
  
    SDO_CS.convert_NTV2_to_XML(ntv2, xml);  
    SDO_CS.convert_XML_to_NTV2(xml, ntv2);  
    DBMS_OUTPUT.PUT_LINE(SUBSTR(ntv2, 1, 32000));  
END;  
/  
NUM_OREC 11
```



```
NUM_SREC 11
NUM_FILE 2
GS_TYPE SECONDS
VERSION NTv2.0
DATUM_F NAD27
DATUM_T NAD83
MAJOR_F 6378206.400
MINOR_F 6356583.800
MAJOR_T 6378137.000
MINOR_T 6356752.314
SUB_NAMEALbanff
PARENT NONE
CREATED 95-06-29
UPDATED 95-07-04
S_LAT 183900.000000
N_LAT 184500.000000
E_LONG 415800.000000
W_LONG 416100.000000
LAT_INC 30.000000
LONG_INC 30.000000
GS_COUNT 231
0.084020 3.737300 0.005000 0.008000
0.083029 3.738740 0.017000 0.011000
0.082038 3.740180 0.029000 0.015000
. . .
```

SDO_CS.CONVERT_XML_TO_NADCON

Format

```
SDO_CS.CONVERT_XML_TO_NADCON(  
    xml_grid IN XMLTYPE,  
    laa_clob OUT CLOB,  
    loa_clob OUT CLOB);
```

Description

Converts an Oracle Spatial XML representation of a NADCON (North American Datum Conversion) grid to NADCON ASCII format.

Parameters

xml_grid

XML document containing the Oracle Spatial XML representation of the NADCON grid.

laa_clob

Output CLOB object containing the latitude values of the NADCON grid.

loa_clob

Output CLOB object containing the longitude values of the NADCON grid.

Usage Notes

To convert a NADCON grid in ASCII format to an Oracle Spatial XML representation, use the [SDO_CS.CONVERT_NADCON_TO_XML](#) procedure.

Examples

The following example converts a NADCON grid in ASCII format to an Oracle Spatial XML representation, converts the resulting XML representation back to a NADCON ASCII representation, and displays the resulting ASCII representation. (Only part of the output is shown.)

```
set lines 32000  
set long 2000000000  
  
DECLARE  
    laa      CLOB;  
    loa      CLOB;  
    xml      XMLTYPE;  
    laa_file BFILE;  
    loa_file BFILE;  
BEGIN  
    laa_file := BFILENAME('MY_WORK_DIR', 'samplenadcon.laa');  
    loa_file := BFILENAME('MY_WORK_DIR', 'samplenadcon.loa');  
    DBMS_LOB.OPEN(laa_file, DBMS_LOB.LOB_READONLY);  
    DBMS_LOB.OPEN(loa_file, DBMS_LOB.LOB_READONLY);  
  
    DBMS_LOB.CREATETEMPORARY(laa, TRUE, DBMS_LOB.SESSION);  
    DBMS_LOB.CREATETEMPORARY(loa, TRUE, DBMS_LOB.SESSION);  
    DBMS_LOB.OPEN(laa, DBMS_LOB.LOB_READWRITE);
```

```

DBMS_LOB.OPEN(lob, DBMS_LOB.LOB_READWRITE);
DBMS_LOB.LOADFROMFILE(lob, lob_file, DBMS_LOB.LOBMAXSIZE);
DBMS_LOB.LOADFROMFILE(lob, lob_file, DBMS_LOB.LOBMAXSIZE);
DBMS_LOB.CLOSE(lob);
DBMS_LOB.CLOSE(lob);
DBMS_LOB.CLOSE(lob_file);
DBMS_LOB.CLOSE(lob_file);

SDO_CS.convert_NADCON_to_XML(lob, lob, xml);
SDO_CS.convert_XML_to_NADCON(xml, lob, lob);
DBMS_OUTPUT.PUT_LINE(SUBSTR(lob, 1, 32000));
DBMS_OUTPUT.PUT_LINE(SUBSTR(lob, 1, 32000));
END;
/
NADCON EXTRACTED REGION                                NADGRD
33 49 1 -107.00000      .25000    25.00000      .25000      .00000
      .006731      .006444      .006208      .006036      .005935      .005904
      .005932      .006002      .006092      .006174      .006218      .006198
      .006087      .005867      .005522      .005045      .004432      .003688
      .002818      .001836      .000759      -.000385      -.001559      -.002704
. . .
NADCON EXTRACTED REGION                                NADGRD
33 49 1 -107.00000      .25000    25.00000      .25000      .00000
      .008509      .007147      .005756      .004331      .002879      .001410
      -.000060      -.001507      -.002904      -.004222      -.005431      -.006498
      -.007395      -.008095      -.008579      -.008832      -.008848      -.008632
      -.008200      -.007577      -.006800      -.005911      -.004957      -.003974
. . .

```

SDO_CS.CONVERT_XML_TO_NTv2

Format

```
SDO_CS.CONVERT_XML_TO_NTv2(
    xml_grid IN XMLTYPE,
    ntv2_clob OUT CLOB);
```

Description

Converts an Oracle Spatial XML representation of an NTv2 (National Transformation Version 2) grid to NTv2 ASCII format.

Parameters

xml_grid

XML document containing the Oracle Spatial XML representation of the NTv2 grid.

ntv2_clob

Output CLOB object containing the values for the NTv2 grid.

Usage Notes

To convert an NTv2 grid in ASCII format to an Oracle Spatial XML representation, use the [SDO_CS.CONVERT_NTv2_TO_XML](#) procedure.

Examples

The following example converts an NTv2 grid in ASCII format to an Oracle Spatial XML representation, converts the resulting XML representation back to an NTv2 ASCII representation, and displays the resulting ASCII representation. (Only part of the output is shown.)

```
set lines 32000
set long 2000000000

DECLARE
    ntv2          CLOB;
    xml           XMLTYPE;
    ntv2_file     BFILE;
BEGIN
    ntv2_file := BFILENAME('MY_WORK_DIR', 'samplentv2.gsa');
    DBMS_LOB.OPEN(ntv2_file, DBMS_LOB.LOB_READONLY);

    DBMS_LOB.CREATETEMPORARY(ntv2, TRUE, DBMS_LOB.SESSION);
    DBMS_LOB.OPEN(ntv2, DBMS_LOB.LOB_READWRITE);
    DBMS_LOB.LOADFROMFILE(ntv2, ntv2_file, DBMS_LOB.LOBMAXSIZE);
    DBMS_LOB.CLOSE(ntv2);
    DBMS_LOB.CLOSE(ntv2_file);

    SDO_CS.convert_NTv2_to_XML(ntv2, xml);
    SDO_CS.convert_XML_to_NTv2(xml, ntv2);
    DBMS_OUTPUT.PUT_LINE(SUBSTR(ntv2, 1, 32000));
END;
/
NUM_OREC 11
NUM_SREC 11
```

```
NUM_FILE 2
GS_TYPE SECONDS
VERSION NTV2.0
DATUM_F NAD27
DATUM_T NAD83
MAJOR_F 6378206.400
MINOR_F 6356583.800
MAJOR_T 6378137.000
MINOR_T 6356752.314
SUB_NAMEALbanff
PARENT NONE
CREATED 95-06-29
UPDATED 95-07-04
S_LAT 183900.000000
N_LAT 184500.000000
E_LONG 415800.000000
W_LONG 416100.000000
LAT_INC 30.000000
LONG_INC 30.000000
GS_COUNT 231
  0.084020 3.737300 0.005000 0.008000
  0.083029 3.738740 0.017000 0.011000
  0.082038 3.740180 0.029000 0.015000
. . .
```

SDO_CS.CREATE_CONCATENATED_OP

Format

```
SDO_CS.CREATE_CONCATENATED_OP(  
    op_id    IN NUMBER,  
    op_name  IN VARCHAR2,  
    use_plan IN TFM_PLAN);
```

Description

Creates a concatenated operation.

Parameters

op_id

ID number of the concatenated operation.

op_name

Name to be associated with the concatenated operation.

use_plan

Transformation plan. The TFM_PLAN object type is explained in [Section 6.5](#).

Usage Notes

A concatenated operation is the concatenation (chaining) of two or more atomic operations.

To create a concatenated operation and make it preferred either systemwide or for a specified use case, you can use the [SDO_CS.CREATE_PREF_CONCATENATED_OP](#) convenience procedure.

Examples

The following example creates a concatenation operation with the operation ID 2999 and the name `CONCATENATED_OPERATION_2999`.

```
DECLARE  
BEGIN  
SDO_CS.CREATE_CONCATENATED_OP(  
    2999,  
    'CONCATENATED_OPERATION_2999',  
    TFM_PLAN(SDO_TFM_CHAIN(4242, 19910, 24200, 1000000000, 24200)));  
END;  
/
```

SDO_CS.CREATE_OBVIOUS_EPSG_RULES

Format

```
SDO_CS.CREATE_OBVIOUS_EPSG_RULES(  
    use_case IN VARCHAR2 DEFAULT NULL);
```

Description

Creates a basic set of EPSG rules to be applied in certain transformations.

Parameters

use_case

Name of the use case to be associated with the application of the EPSG rules that are created. Must be a value from the USE_CASE column of the SDO_PREFERRED_OPS_USER table (described in [Section 6.6.25](#)).

Usage Notes

This procedure creates rules to implement the main EPSG-defined transformations between specific coordinate reference systems. For transformations between some coordinate reference systems, EPSG has specified rules that should be applied. For any given transformation from one coordinate reference system to another, the EPSG rule might be different from the default Oracle Spatial rule. If you execute this procedure, the EPSG rules are applied in any such cases. If you do not execute this procedure, the default Spatial rules are used in such cases.

This procedure inserts many rows into the SDO_PREFERRED_OPS_SYSTEM table (see [Section 6.6.24](#)).

To delete the EPSG rules created by this procedure, and thus cause the default Spatial rules to be used in all cases, use the [SDO_CS.DELETE_ALL_EPSG_RULES](#) procedure.

Examples

The following example creates a basic set of EPSG rules to be applied in certain transformations.

```
EXECUTE SDO_CS.CREATE_OBVIOUS_EPSG_RULES;
```

SDO_CS.CREATE_PREF_CONCATENATED_OP

Format

```
SDO_CS.CREATE_PREF_CONCATENATED_OP(  
    op_id    IN NUMBER,  
    op_name  IN VARCHAR2,  
    use_plan IN TFM_PLAN,  
    use_case IN VARCHAR2 DEFAULT NULL);
```

Description

Creates a concatenated operation, associating it with a transformation plan and making it preferred either systemwide or for a specified use case.

Parameters

op_id

ID number of the concatenated operation to be created.

op_name

Name to be associated with the concatenated operation.

use_plan

Transformation plan. The TFM_PLAN object type is explained in [Section 6.5](#).

use_case

Use case to which this preferred concatenated operation applies. Must be a null or a value from the USE_CASE column of the SDO_PREFERRED_OPS_USER table (described in [Section 6.6.25](#)).

Usage Notes

This convenience procedure combines the operations of the [SDO_CS.CREATE_CONCATENATED_OP](#) and [SDO_CS.ADD_PREFERENCE_FOR_OP](#) procedures.

A concatenated operation is the concatenation (chaining) of two or more atomic operations.

If `use_case` is null, the transformation plan associated with the operation is a systemwide preference, and a row is added (or two rows are added if a reverse operation exists) to the SDO_PREFERRED_OPS_SYSTEM table (described in [Section 6.6.24](#)). If `use_case` is not null, the transformation plan associated with the operation is a preference associated with the specified use case, and a row is added (or two rows are added if a reverse operation exists) to the SDO_PREFERRED_OPS_USER table (described in [Section 6.6.25](#)).

To create a concatenation without making it preferred either systemwide or for a specified use case, use the [SDO_CS.CREATE_CONCATENATED_OP](#) procedure

To delete a concatenated operation, use the [SDO_CS.DELETE_OP](#) procedure.

Examples

The following example creates a concatenation operation with the operation ID 300 and the name MY_CONCATENATION_OPERATION, and causes Spatial to use the

specified transformation plan in all cases (because use_case is null) when this operation is used.

```
DECLARE
BEGIN
SDO_CS.CREATE_PREF_CONCATENATED_OP(
  300,
  'MY_CONCATENATED_OPERATION',
  TFM_PLAN(SDO_TFM_CHAIN(4242, 19910, 24200, 1000000000, 24200)),
  NULL);
END;
/
```

SDO_CS.DELETE_ALL_EPSG_RULES

Format

```
SDO_CS.DELETE_ALL_EPSG_RULES(  
    use_case IN VARCHAR2 DEFAULT NULL);
```

Description

Deletes the basic set of EPSG rules to be applied in certain transformations.

Parameters

use_case

Name of the use case to be associated with the application of the EPSG rules that are created. Must match the value that was used for the `use_case` parameter value (either null or a specified value) when the [SDO_CS.CREATE_OBVIOUS_EPSG_RULES](#) procedure was called.

Usage Notes

This procedure deletes the EPSG rules that were previously created by the [SDO_CS.CREATE_OBVIOUS_EPSG_RULES](#) procedure, and thus causes the default Spatial rules to be used in all cases. (See the Usage Notes for the [SDO_CS.CREATE_OBVIOUS_EPSG_RULES](#) procedure for more information.)

If `use_case` is null, this procedure deletes all rows from the `SDO_PREFERRED_OPS_SYSTEM` table (see [Section 6.6.24](#)). If `use_case` is not null, this procedure deletes the rows associated with the specified use case from the `SDO_PREFERRED_OPS_USER` table (see [Section 6.6.25](#)).

Examples

The following example deletes the basic set of EPSG rules to be applied in certain transformations.

```
EXECUTE SDO_CS.DELETE_ALL_EPSG_RULES;
```

SDO_CS.DELETE_OP

Format

```
SDO_CS.DELETE_OP(  
    op_id IN NUMBER);
```

Description

Deletes a concatenated operation.

Parameters

op_id
ID number of the operation to be deleted.

Usage Notes

To create a concatenated operation and make it preferred systemwide or only for a specified use case, use the [SDO_CS.CREATE_CONCATENATED_OP](#) procedure.

Examples

The following example deletes the operation with the ID number 300.

```
EXECUTE SDO_CS.DELETE_OP(300);
```

SDO_CS.DETERMINE_CHAIN

Format

```
SDO_CS.DETERMINE_CHAIN(
    transient_rule_set IN SDO_TRANSIENT_RULE_SET,
    use_case           IN VARCHAR2,
    source_srid        IN NUMBER,
    target_srid        IN NUMBER) RETURN TFM_PLAN;
```

Description

Returns the query chain, based on the system rule set, to be used in transformations from one coordinate reference system to another coordinate reference system.

Parameters

transient_rule_set

Rule set to be used for the transformation. If you specify a null value, the Oracle system rule set is used.

use_case

Use case for which to determine the query chain. Must be a null value or a value from the USE_CASE column of the SDO_PREFERRED_OPS_USER table (described in [Section 6.6.25](#)).

source_srid

The SRID of the source coordinate reference system. Must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

target_srid

The SRID of the target coordinate reference system. Must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

Usage Notes

This function returns an object of type TFM_PLAN, which is explained in [Section 6.5](#).

The `transient_rule_set` parameter is of type SDO_TRANSIENT_RULE_SET, which has the following definition:

```
CREATE TYPE sdo_transient_rule_set AS OBJECT (
    source_srid NUMBER,
    target_srid NUMBER,
    tfm          NUMBER);
```

Examples

The following example returns the query chain based on the system rule set.

```
SELECT MDSYS.SDO_CS.DETERMINE_CHAIN(NULL, NULL, 4804, 4257) FROM DUAL;
```

```
MDSYS.SDO_CS.DETERMINE_CHAIN(NULL, NULL, 4804, 4257) (THE_PLAN)
```

```
-----
TFM_PLAN(SDO_TFM_CHAIN(4804, -2, 4257))
```

The next example creates a preferred concatenated operation (with operation ID 300) with a specified chain for transformations from SRID 4804 to SRID 4257, and then calls the DETERMINE_CHAIN function, returning a different result. (The operation created in this example is not meaningful or useful, and it was created only for illustration.)

```

CALL SDO_CS.CREATE_PREF_CONCATENATED_OP(
  300,
  'CONCATENATED OPERATION',
  TFM_PLAN(
    SDO_TFM_CHAIN(
      4804,
      1000000001, 4804,
      1000000002, 4804,
      1000000001, 4804,
      1000000001, 4804,
      1000000002, 4804,
      1000000002, 4804,
      1000000002, 4804,
      1000000001, 4804,
      1000000001, 4804,
      1000000001, 4804,
      1000000001, 4804,
      1000000002, 4804,
      1000000002, 4804,
      1000000002, 4257)),
  NULL);

SELECT MDSYS.SDO_CS.DETERMINE_CHAIN(NULL, NULL, 4804, 4257) FROM DUAL;

MDSYS.SDO_CS.DETERMINE_CHAIN(NULL,NULL,4804,4257) (THE_PLAN)
-----
TFM_PLAN(SDO_TFM_CHAIN(4804, 300, 4257))

```

SDO_CS.DETERMINE_DEFAULT_CHAIN

Format

```
SDO_CS.DETERMINE_DEFAULT_CHAIN(  
    source_srid IN NUMBER,  
    target_srid IN NUMBER) RETURN SDO_SRID_CHAIN;
```

Description

Returns the default chain of SRID values in transformations from one coordinate reference system to another coordinate reference system.

Parameters

source_srid

The SRID of the source coordinate reference system. Must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

target_srid

The SRID of the target coordinate reference system. Must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

Usage Notes

This function returns an object of type SDO_SRID_CHAIN, which is defined as VARRAY(1048576) OF NUMBER.

Examples

The following example returns the default chain of SRID values in transformations from SRID 4804 to SRID 4257.

```
SELECT MDSYS.SDO_CS.DETERMINE_DEFAULT_CHAIN(4804, 4257) FROM DUAL;  
  
MDSYS.SDO_CS.DETERMINE_DEFAULT_CHAIN(4804,4257)  
-----  
SDO_SRID_CHAIN(NULL, 4804, 4257, NULL)
```

SDO_CS.FIND_GEOG_CRS

Format

```
SDO_CS.FIND_GEOG_CRS(
    reference_srid      IN NUMBER,
    is_legacy           IN VARCHAR2,
    max_rel_num_difference IN NUMBER DEFAULT 0.000001) RETURN SDO_SRID_LIST;
```

Description

Returns the SRID values of geodetic (geographic) coordinate reference systems that have the same well-known text (WKT) numeric values as the coordinate reference system with the specified reference SRID value.

Parameters

reference_srid

The SRID of the coordinate reference system for which to find all other geodetic coordinate reference systems that have the same WKT numeric values. Must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

is_legacy

TRUE limits the results to geodetic coordinate reference systems for which the IS_LEGACY column value is TRUE in the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)); FALSE limits the results to geodetic coordinate reference systems for which the IS_LEGACY column value is FALSE in the SDO_COORD_REF_SYS table. If you specify a null value for this parameter, the IS_LEGACY column value in the SDO_COORD_REF_SYS table is ignored in determining the results.

max_rel_num_difference

A numeric value indicating how closely WKT values must match in order for a projected coordinate reference system to be considered a match. The default value is 0.000001. The value for each numeric WKT item is compared with its corresponding value in the WKT for the reference SRID or in the specified list of parameters to this function; and if the difference in all cases is less than or equal to the max_rel_num_difference value, the SRID for that coordinate reference system is included in the results.

Usage Notes

This function returns an object of type SDO_SRID_LIST, which is defined as VARRAY(1048576) OF NUMBER.

The well-known text (WKT) format is described in [Section 6.7.1.1](#).

Examples

The following examples show the effect of the is_legacy parameter value on the results. The first example returns the SRID values of all geodetic legacy coordinate reference systems that have the same WKT numeric values as the coordinate reference system with the SRID value of 8307.

```
SELECT SDO_CS.FIND_GEOG_CRS (
```

```
8307,
TRUE') FROM DUAL;
```

```
SDO_CS.FIND_GEOG_CRS(8307,'TRUE')
```

```
-----
SDO_SRID_LIST(8192, 8265, 8307, 8311, 8320, 524288, 2000002, 2000006, 2000012, 2000015, 2000023, 2000028)
```

The next example returns the SRID values of all geodetic non-legacy coordinate reference systems that have the same WKT numeric values as the coordinate reference system with the SRID value of 8307.

```
SELECT SDO_CS.FIND_GEOG_CRS(
8307,
FALSE') FROM DUAL;
```

```
SDO_CS.FIND_GEOG_CRS(8307,'FALSE')
```

```
-----
SDO_SRID_LIST(4019, 4030, 4031, 4032, 4033, 4041, 4121, 4122, 4126, 4130, 4133, 4140, 4141, 4148, 4151, 4152, 4163, 4166, 4167, 4170, 4171, 4172, 4173, 4176, 4180, 4189, 4190, 4258, 4269, 4283, 4318, 4319, 4326, 4610, 4612, 4617, 4619, 4624, 4627, 4640, 4659, 4661, 4667, 4669, 4670)
```

The next example returns the SRID values of all geodetic coordinate reference systems (legacy and non-legacy) that have the same WKT numeric values as the coordinate reference system with the SRID value of 8307.

```
SELECT SDO_CS.FIND_GEOG_CRS(
8307,
NULL) FROM DUAL;
```

```
SDO_CS.FIND_GEOG_CRS(8307,NULL)
```

```
-----
SDO_SRID_LIST(4019, 4030, 4031, 4032, 4033, 4041, 4121, 4122, 4126, 4130, 4133, 4140, 4141, 4148, 4151, 4152, 4163, 4166, 4167, 4170, 4171, 4172, 4173, 4176, 4180, 4189, 4190, 4258, 4269, 4283, 4318, 4319, 4326, 4610, 4612, 4617, 4619, 4624, 4627, 4640, 4659, 4661, 4667, 4669, 4670, 8192, 8265, 8307, 8311, 8320, 524288, 2000002, 2000006, 2000012, 2000015, 2000023, 2000028)
```

SDO_CS.FIND_PROJ_CRS

Format

```
SDO_CS.FIND_PROJ_CRS(
    reference_srid      IN NUMBER,
    is_legacy           IN VARCHAR2,
    max_rel_num_difference IN NUMBER DEFAULT 0.000001) RETURN SDO_SRID_LIST;
```

Description

Returns the SRID values of projected coordinate reference systems that have the same well-known text (WKT) numeric values as the coordinate reference system with the specified reference SRID value.

Parameters

reference_srid

The SRID of the coordinate reference system for which to find all other projected coordinate reference systems that have the same WKT numeric values. Must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

is_legacy

TRUE limits the results to projected coordinate reference systems for which the IS_LEGACY column value is TRUE in the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)); FALSE limits the results to projected coordinate reference systems for which the IS_LEGACY column value is FALSE in the SDO_COORD_REF_SYS table. If you specify a null value for this parameter, the IS_LEGACY column value in the SDO_COORD_REF_SYS table is ignored in determining the results.

max_rel_num_difference

A numeric value indicating how closely WKT values must match in order for a coordinate reference system to be considered a match. The default value is 0.000001. The value for each numeric WKT item is compared with its corresponding value in the WKT for the reference SRID or in the specified list of parameters to this function; and if the difference in all cases is less than or equal to the max_rel_num_difference value, the SRID for that coordinate reference system is included in the results.

Usage Notes

This function returns an object of type SDO_SRID_LIST, which is defined as VARRAY(1048576) OF NUMBER.

The well-known text (WKT) format is described in [Section 6.7.1.1](#).

Examples

The following examples show the effect of the is_legacy parameter value on the results. The first example returns the SRID values of all projected legacy coordinate reference systems that have the same WKT numeric values as the coordinate reference system with the SRID value of 2007. The returned result list is empty, because there are no legacy projected legacy coordinate reference systems that meet the search criteria.

```
SELECT SDO_CS.FIND_PROJ_CRS(
```

```
2007,  
'TRUE') FROM DUAL;
```

```
SDO_CS.FIND_PROJ_CRS(2007, 'TRUE')
```

```
-----  
SDO_SRID_LIST()
```

The next example returns the SRID values of all projected non-legacy coordinate reference systems that have the same WKT numeric values as the coordinate reference system with the SRID value of 2007.

```
SELECT SDO_CS.FIND_PROJ_CRS(  
2007,  
'FALSE') FROM DUAL;
```

```
SDO_CS.FIND_PROJ_CRS(2007, 'FALSE')
```

```
-----  
SDO_SRID_LIST(2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 21291)
```

The next example returns the SRID values of all projected coordinate reference systems (legacy and non-legacy) that have the same WKT numeric values as the coordinate reference system with the SRID value of 2007. The returned result list is the same as for the preceding example.

```
SELECT SDO_CS.FIND_PROJ_CRS(  
2007,  
NULL) FROM DUAL;
```

```
SDO_CS.FIND_PROJ_CRS(2007, NULL)
```

```
-----  
SDO_SRID_LIST(2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 21291)
```

SDO_CS.FROM_OGC_SIMPLEFEATURE_SRS

Format

```
SDO_CS.FROM_OGC_SIMPLEFEATURE_SRS(
    wkt IN VARCHAR2) RETURN VARCHAR2;
```

Description

Converts a well-known text string from the Open Geospatial Consortium simple feature format without the TOWGS84 keyword to the format that includes the TOWGS84 keyword.

Parameters

wkt
Well-known text string.

Usage Notes

To convert a well-known text string from the Open Geospatial Consortium simple feature format that includes the TOWGS84 keyword to the format without the TOWGS84 keyword, use the [SDO_CS.TO_OGC_SIMPLEFEATURE_SRS](#) function.

Examples

The following example converts a well-known text string from the Open Geospatial Consortium simple feature format without the TOWGS84 keyword to the format that includes the TOWGS84 keyword.

```
SELECT sdo_cs.from_ogc_simplefeature_srs('GEOGCS [ "Longitude / Latitude (DHDN)",
    DATUM [ "", SPHEROID [ "Bessel 1841", 6377397.155, 299.1528128],
    582.000000, 105.000000, 414.000000, -1.040000, -0.350000, 3.080000, 8.300000 ],
    PRIMEM [ "Greenwich", 0.000000 ], UNIT [ "Decimal Degree",
0.01745329251994330]')
FROM DUAL;
```

```
MDSYS.SDO_CS.FROM_OGC_SIMPLEFEATURE_SRS('GEOGCS["LONGITUDE/LATITUDE(DHDN)",DATUM
-----
GEOGCS [ "Longitude / Latitude (DHDN)", DATUM [ "", SPHEROID [ "Bessel 1841", 6377
397.155, 299.1528128], TOWGS84[ 582.000000, 105.000000, 414.000000, -1.040000, -
0.350000, 3.080000, 8.300000]], PRIMEM [ "Greenwich", 0.000000 ], UNIT [ "Decimal
Degree", 0.01745329251994330])
```

SDO_CS.FROM_USNG

Format

```
SDO_CS.FROM_USNG(  
    usng IN VARCHAR2,  
    srid IN NUMBER,  
    datum IN VARCHAR2 DEFAULT 'NAD83') RETURN SDO_GEOMETRY;
```

Description

Converts a point represented in U.S. National Grid format to a spatial point geometry object.

Parameters

usng

Well-known text string.

srid

The SRID of the coordinate system to be used for the conversion (that is, the SRID to be used in the returned geometry). Must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

datum

The name of the datum on which the U.S. National Grid coordinate for the point is based. Must be either a value in the DATUM_NAME column of the SDO_DATUMS table (described in [Section 6.6.22](#)) or null. The default value is NAD83.

Usage Notes

For information about Oracle Spatial support for the U.S. National Grid, see [Section 6.10](#).

To convert a spatial point geometry to a point represented in U.S. National Grid format, use the [SDO_CS.TO_USNG](#) function.

Examples

The following example converts a point represented in U.S. National Grid format to a spatial geometry point object with longitude/latitude coordinates.

```
-- Convert US National Grid point to SDO_GEMETRY point using SRID 4326  
-- (WGS 84, longitude/latitude).  
SELECT SDO_CS.FROM_USNG(  
    '18SUJ2348316806479498',  
    4326) FROM DUAL;  
  
WGS84(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)  
-----  
SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(-77.03524, 38.8894673, NULL), NULL, NULL)
```

SDO_CS.MAP_EPSG_SRID_TO_ORACLE

Format

```
SDO_CS.MAP_EPSG_SRID_TO_ORACLE(  
    epsg_srid IN NUMBER) RETURN NUMBER;
```

Description

Returns the Oracle Spatial SRID value corresponding to the specified EPSG SRID value.

Parameters

epsg_srid

The SRID of the EPSG coordinate reference system, as indicated in the COORD_REF_SYS_CODE field in the EPSG Coordinate Reference System table.

Usage Notes

This function returns a value that matches a value in the SRID column of the SDO_COORD_REF_SYS table (see [Section 6.6.9](#)).

To return the EPSG SRID value corresponding to the specified Oracle Spatial SRID value, use the [SDO_CS.MAP_ORACLE_SRID_TO_EPSG](#) function.

Examples

The following example returns the Oracle Spatial SRID value corresponding to EPSG SRID 23038.

```
SELECT SDO_CS.MAP_EPSG_SRID_TO_ORACLE(23038) FROM DUAL;
```

```
SDO_CS.MAP_EPSG_SRID_TO_ORACLE(23038)  
-----  
                                82361
```

SDO_CS.MAP_ORACLE_SRID_TO_EPSG

Format

```
SDO_CS.MAP_ORACLE_SRID_TO_EPSG(  
    legacy_srid IN NUMBER) RETURN NUMBER;
```

Description

Returns the EPSG SRID value corresponding to the specified Oracle Spatial SRID value.

Parameters

legacy_srid

Oracle Spatial SRID value. Must match a value in the LEGACY_CODE column of the SDO_COORD_REF_SYS table (see [Section 6.6.9](#)).

Usage Notes

This function returns the SRID of an EPSG coordinate reference system. The EPSG SRID value for a coordinate reference system is indicated in the COORD_REF_SYS_CODE field in the EPSG Coordinate Reference System table.

To return the Oracle Spatial SRID value corresponding to a specified EPSG SRID value, use the [SDO_CS.MAP_EPSG_SRID_TO_ORACLE](#) function.

Examples

The following example returns the EPSG SRID value corresponding to Oracle Spatial SRID 82361.

```
SELECT SDO_CS.MAP_ORACLE_SRID_TO_EPSG(82361) FROM DUAL;
```

```
SDO_CS.MAP_ORACLE_SRID_TO_EPSG(82361)  
-----  
                                23038
```

SDO_CS.REVOKE_PREFERENCE_FOR_OP

Format

```
SDO_CS.REVOKE_PREFERENCE_FOR_OP(  
    op_id      IN NUMBER,  
    source_crs IN NUMBER DEFAULT NULL,  
    target_crs IN NUMBER DEFAULT NULL,  
    use_case   IN VARCHAR2 DEFAULT NULL);
```

Description

Revokes a preference for an operation between a source coordinate system and a target coordinate system.

Parameters

op_id

ID number of the operation. Must match an `op_id` value that was specified in a call to the [SDO_CS.ADD_PREFERENCE_FOR_OP](#) procedure.

source_crs

The SRID of the source coordinate reference system. Must match the `source_crs` value in a `source_crs`, `target_crs`, and `use_case` combination that was specified in a call to the [SDO_CS.ADD_PREFERENCE_FOR_OP](#) procedure.

target_crs

The SRID of the target coordinate reference system. Must match the `target_crs` value in a `source_crs`, `target_crs`, and `use_case` combination that was specified in a call to the [SDO_CS.ADD_PREFERENCE_FOR_OP](#) procedure.

use_case

Name of the use case associated with the preference. Must match the `use_case` value in a `source_crs`, `target_crs`, and `use_case` combination that was specified in a call to the [SDO_CS.ADD_PREFERENCE_FOR_OP](#) procedure.

Usage Notes

This procedure reverses the effect of the [SDO_CS.ADD_PREFERENCE_FOR_OP](#) procedure.

If `use_case` is null, this procedure deletes one or more rows from the `SDO_PREFERRED_OPS_SYSTEM` table (described in [Section 6.6.24](#)). If `use_case` is not null, this procedure deletes one or more rows from the `SDO_PREFERRED_OPS_USER` table (described in [Section 6.6.25](#)).

Examples

The following example revokes a preference for operation ID 19777 to be used in transformations from SRID 4301 to SRID 4326 when use case `use_case_B` is specified for the transformation.

```
EXECUTE SDO_CS.REVOKE_PREFERENCE_FOR_OP(19777, 4301, 4326, 'use_case_B');
```

SDO_CS.TO_OGC_SIMPLEFEATURE_SRS

Format

```
SDO_CS.TO_OGC_SIMPLEFEATURE_SRS(  
    wkt IN VARCHAR2) RETURN VARCHAR2;
```

Description

Converts a well-known text string from the Open Geospatial Consortium simple feature format that includes the TOWGS84 keyword to the format without the TOWGS84 keyword.

Parameters

wkt
Well-known text string.

Usage Notes

To convert a well-known text string from the Open Geospatial Consortium simple feature format without the TOWGS84 keyword to the format that includes the TOWGS84 keyword, use the [SDO_CS.FROM_OGC_SIMPLEFEATURE_SRS](#) procedure.

Examples

The following example converts a well-known text string from the Open Geospatial Consortium simple feature format that includes the TOWGS84 keyword to the format without the TOWGS84 keyword.

```
SELECT sdo_cs.to_ogc_simplefeature_srs('GEOGCS [ "Longitude / Latitude (DHDN)",  
    DATUM [ "", SPHEROID [ "Bessel 1841", 6377397.155, 299.1528128 ],  
    TOWGS84 [ 582.000000, 105.000000, 414.000000, -1.040000, -0.350000, 3.080000,  
    8.300000 ] ],  
    PRIMEM [ "Greenwich", 0.000000 ], UNIT [ "Decimal Degree",  
    0.01745329251994330 ]')  
FROM DUAL;
```

```
MDSYS.SDO_CS.TO_OGC_SIMPLEFEATURE_SRS('GEOGCS["LONGITUDE/LATITUDE (DHDN)",DATUM["  
-----  
GEOGCS [ "Longitude / Latitude (DHDN)", DATUM [ "", SPHEROID [ "Bessel 1841", 6377  
397.155, 299.1528128 ], 582.000000, 105.000000, 414.000000, -1.040000, -0.350000,  
3.080000, 8.300000 ], PRIMEM [ "Greenwich", 0.000000 ], UNIT [ "Decimal Degree",  
0.01745329251994330 ]]
```

SDO_CS.TO_USNG

Format

```
SDO_CS.TO_USNG(  
    geom           IN SDO_GEOMETRY,  
    accuracy_in_meters IN NUMBER,  
    datum         IN VARCHAR2 DEFAULT 'NAD83') RETURN VARCHAR2;
```

Description

Converts a spatial point geometry object to a point represented in U.S. National Grid format.

Parameters

geom

Point geometry whose representation is to be converted to a point represented in U.S. National Grid format. The input geometry must have a valid non-null SRID, that is, a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

accuracy_in_meters

Accuracy of the point location in meters. Should be 1 raised to a negative or positive power of 10 (for example, 0.001, 0.01, 0.1, 1, 10, 100, or 1000). Any other specified values are adjusted internally by Spatial, and the result might not be what you expect.

datum

The name of the datum on which the U.S. National Grid coordinate for the point is to be based. Must be either NAD83 or NAD27. The default value is NAD83.

Usage Notes

For information about Oracle Spatial support for the U.S. National Grid, see [Section 6.10](#).

The `accuracy_in_meters` value affects the number of digits used to represent the accuracy in the returned U.S. National Grid string. For example, if you specify 0.000001, the string will contain many digits; however, depending on the source of the data, the digits might not accurately reflect geographical reality. Consider the following scenarios. If you create a U.S. National Grid string from a UTM geometry, you can get perfect accuracy, because no inherently inaccurate transformation is involved. However, transforming from a Lambert projection to the U.S. National Grid format involves an inverse Lambert projection and a forward UTM projection, each of which has some inherent inaccuracy. If you request the resulting U.S. National Grid string with 1 millimeter (0.001) accuracy, the string will contain all the digits, but the millimeter-level digit will probably be geographically inaccurate.

To convert a point represented in U.S. National Grid format to a spatial point geometry, use the [SDO_CS.FROM_USNG](#) function.

Examples

The following example converts a spatial geometry point object with longitude/latitude coordinates to a point represented in U.S. National Grid format using an accuracy of 0.001 meter (1 millimeter).

```
-- Convert longitude/latitude (WGS 84) point to US National Grid.
SELECT SDO_CS.TO_USNG(
  SDO_GEOMETRY(2001, 4326,
    SDO_POINT_TYPE(-77.0352402158258, 38.8894673086544, NULL),
    NULL, NULL),
  0.001) FROM DUAL;

SDO_CS.TO_USNG(SDO_GEOMETRY(2001,4326,SDO_POINT_TYPE(-77.0352402158258,38.889467
-----
18SUJ2348316806479498
```

SDO_CS.TRANSFORM

Format

```
SDO_CS.TRANSFORM(  
    geom IN SDO_GEOMETRY,  
    to_srid IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_CS.TRANSFORM(  
    geom IN SDO_GEOMETRY,  
    tolerance IN NUMBER,  
    to_srid IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_CS.TRANSFORM(  
    geom IN SDO_GEOMETRY,  
    dim IN SDO_DIM_ARRAY,  
    to_srid IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_CS.TRANSFORM(  
    geom IN SDO_GEOMETRY,  
    to_srsname IN VARCHAR2  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_CS.TRANSFORM(  
    geom IN SDO_GEOMETRY,  
    tolerance IN NUMBER,  
    to_srsname IN VARCHAR2  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_CS.TRANSFORM(  
    geom IN SDO_GEOMETRY,  
    dim IN SDO_DIM_ARRAY,  
    to_srsname IN VARCHAR2  
    ) RETURN SDO_GEOMETRY;
```

Description

Transforms a geometry representation using a coordinate system (specified by SRID or name).

Parameters

geom

Geometry whose representation is to be transformed using another coordinate system. The input geometry must have a valid non-null SRID, that is, a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

tolerance

Tolerance value (see [Section 1.5.5](#)).

dim

Dimensional information array corresponding to geom, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in [Section 2.6](#)).

to_srid

The SRID of the coordinate system to be used for the transformation. It must be a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

to_sname

The name of the coordinate system to be used for the transformation. It must be a value (specified exactly) in the COORD_REF_SYS_NAME column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

Usage Notes

Transformation can be done only between two different georeferenced coordinate systems or between two different local coordinate systems.

An exception is raised if geom, to_srid, or to_sname is invalid. For geom to be valid for this function, its definition must include an SRID value matching a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

Examples

The following example transforms the cola_c geometry to a representation that uses SRID value 8199. (This example uses the definitions from the example in [Section 6.11](#).)

```
-- Return the transformation of cola_c using to_srid 8199
-- ('Longitude / Latitude (Arc 1950)')
SELECT c.name, SDO_CS.TRANSFORM(c.shape, m.diminfo, 8199)
  FROM cola_markets_cs c, user_sdo_geom_metadata m
  WHERE m.table_name = 'COLA_MARKETS_CS' AND m.column_name = 'SHAPE'
  AND c.name = 'cola_c';

NAME
-----
SDO_CS.TRANSFORM(C.SHAPE,M.DIMINFO,8199) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z)
-----
cola_c
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(3.00074114, 3.00291482, 6.00067068, 3.00291287, 6.0006723, 5.00307625, 4.0007
1961, 5.00307838, 3.00074114, 3.00291482))

-- Same as preceding, but using to_sname parameter.
```

```
SELECT c.name, SDO_CS.TRANSFORM(c.shape, m.diminfo,  
    'Longitude / Latitude (Arc 1950)')  
FROM cola_markets_cs c, user_sdo_geom_metadata m  
WHERE m.table_name = 'COLA_MARKETS_CS' AND m.column_name = 'SHAPE'  
AND c.name = 'cola_c';
```

NAME

```
-----  
SDO_CS.TRANSFORM(C.SHAPE,M.DIMINFO,'LONGITUDE/LATITUDE(ARC1950)')(SDO_GTYPE, SDO  
-----  
cola_c  
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR  
AY(3.00074114, 3.00291482, 6.00067068, 3.00291287, 6.0006723, 5.00307625, 4.0007  
1961, 5.00307838, 3.00074114, 3.00291482))
```

SDO_CS.TRANSFORM_LAYER

Format

```
SDO_CS.TRANSFORM_LAYER(  
    table_in   IN VARCHAR2,  
    column_in  IN VARCHAR2,  
    table_out  IN VARCHAR2,  
    to_srid    IN NUMBER);
```

or

```
SDO_CS.TRANSFORM_LAYER(  
    table_in   IN VARCHAR2,  
    column_in  IN VARCHAR2,  
    table_out  IN VARCHAR2,  
    use_plan   IN TFM_PLAN);
```

or

```
SDO_CS.TRANSFORM_LAYER(  
    table_in   IN VARCHAR2,  
    column_in  IN VARCHAR2,  
    table_out  IN VARCHAR2,  
    use_case   IN VARCHAR2,  
    to_srid    IN NUMBER);
```

Description

Transforms an entire layer of geometries (that is, all geometries in a specified column in a table).

Parameters

table_in

Table containing the layer (`column_in`) whose geometries are to be transformed.

column_in

Column in `table_in` that contains the geometries to be transformed.

table_out

Table that will be created and that will contain the results of the transformation. See the Usage Notes for information about the format of this table.

to_srid

The SRID of the coordinate system to be used for the transformation. `to_srid` must be a value in the SRID column of the `SDO_COORD_REF_SYS` table (described in [Section 6.6.9](#)).

use_plan

Transformation plan. The TFM_PLAN object type is explained in [Section 6.5](#).

use_case

Name of the use case whose transformation rules are to be applied in performing the transformation. Use cases are explained in [Section 6.4](#).

Usage Notes

Transformation can be done only between two different georeferenced coordinate systems or between two different local coordinate systems.

An exception is raised if any of the following occurs:

- `table_in` does not exist, or `column_in` does not exist in the table.
- The geometries in `column_in` have a null or invalid SDO_SRID value.
- `table_out` already exists.
- `to_srid` is invalid.

The `table_out` table is created by the procedure and is filled with one row for each transformed geometry. This table has the columns shown in [Table 13–2](#).

Table 13–2 Table to Hold Transformed Layer

Column Name	Data Type	Description
SDO_ROWID	ROWID	Oracle ROWID (row address identifier). For more information about the ROWID data type, see <i>Oracle Database SQL Reference</i> .
GEOMETRY	SDO_GEOMETRY	Geometry object with coordinate values in the specified (<code>to_srid</code> parameter) coordinate system.

Examples

The following example transforms the geometries in the `shape` column in the `COLA_MARKETS_CS` table to a representation that uses SRID value 8199. The transformed geometries are stored in the newly created table named `COLA_MARKETS_CS_8199`. (This example uses the definitions from the example in [Section 6.11](#).)

```
-- Transform the entire SHAPE layer and put results in the table
-- named cola_markets_cs_8199, which the procedure will create.
CALL SDO_CS.TRANSFORM_LAYER('COLA_MARKETS_CS','SHAPE','COLA_MARKETS_CS_8199',8199);
```

[Example 6–8](#) in [Section 6.11](#) includes a display of the geometry object coordinates in both tables (`COLA_MARKETS_CS` and `COLA_MARKETS_CS_8199`).

SDO_CS.UPDATE_WKTS_FOR_ALL_EPSG_CRS

Format

```
SDO_CS.UPDATE_WKTS_FOR_ALL_EPSG_CRS();
```

Description

Updates the well-known text (WKT) description for all EPSG coordinate reference systems.

Parameters

None.

Usage Notes

For information about using procedures to update well-known text (WKT) description, see [Section 6.7.1.2](#).

Examples

The following example updates the WKT description for all EPSG coordinate reference systems.

```
EXECUTE SDO_CS.UPDATE_WKTS_FOR_ALL_EPSG_CRS;  
Updating SRID 4001...  
Updating SRID 4002...  
Updating SRID 4003...  
...  
Updating SRID 69036405...  
Updating SRID 69046405...
```


SDO_CS.UPDATE_WKTS_FOR_EPSG_CRS

Format

```
SDO_CS.UPDATE_WKTS_FOR_EPSG_CRS(  
    srid IN NUMBER);
```

Description

Updates the well-known text (WKT) description for the EPSG coordinate reference system associated with a specified SRID.

Parameters

srid

The SRID of the coordinate system whose well-known text (WKT) description is to be updated. An entry for the specified value must exist in the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

Usage Notes

For information about using procedures to update well-known text (WKT) description, see [Section 6.7.1.2](#).

Examples

The following example updates the WKT description for the EPSG coordinate reference system associated with SRID 8307.

```
EXECUTE SDO_CS.UPDATE_WKTS_FOR_EPSG_CRS(8307);
```

SDO_CS.UPDATE_WKTS_FOR_EPSG_DATUM

Format

```
SDO_CS.UPDATE_WKTS_FOR_EPSG_DATUM(  
    datum_id IN NUMBER);
```

Description

Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified datum.

Parameters

datum_id

The ID of the datum. Must match a value in the DATUM_ID column of the SDO_DATUMS table (described in [Section 6.6.22](#)).

Usage Notes

For information about using procedures to update well-known text (WKT) description, see [Section 6.7.1.2](#).

Examples

The following example updates the WKT description for all EPSG coordinate reference systems associated with datum 5100.

```
EXECUTE SDO_CS.UPDATE_WKTS_FOR_EPSG_DATUM(5100);  
Updating SRID 5714...  
Updating SRID 5715...
```

SDO_CS.UPDATE_WKTS_FOR_EPSG_ELLIPS

Format

```
SDO_CS.UPDATE_WKTS_FOR_EPSG_ELLIPS(  
    ellipsoid_id IN NUMBER);
```

Description

Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified ellipsoid.

Parameters

ellipsoid_id

The ID of the ellipsoid. Must match a value in the ELLIPSOID_ID column of the SDO_ELLIPSOIDS table (described in [Section 6.6.23](#)).

Usage Notes

For information about using procedures to update well-known text (WKT) description, see [Section 6.7.1.2](#).

Examples

The following example updates the WKT description for all EPSG coordinate reference systems associated with ellipsoid 7100.

```
EXECUTE SDO_CS.UPDATE_WKTS_FOR_EPSG_ELLIPS(7001);  
Updating SRID 4001...  
Updating SRID 4188...  
Updating SRID 29901...  
Updating SRID 61886405...  
Updating SRID 4277...  
Updating SRID 27700...  
Updating SRID 62776405...  
Updating SRID 4278...  
Updating SRID 62786405...  
Updating SRID 4279...  
Updating SRID 62796405...
```

SDO_CS.UPDATE_WKTS_FOR_EPSG_OP

Format

```
SDO_CS.UPDATE_WKTS_FOR_EPSG_OP(  
    coord_op_id IN NUMBER);
```

Description

Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified coordinate transformation operation.

Parameters

coord_op_id

The ID of the SRID of the coordinate transformation operation. Must match a value in the COORD_OP_ID column of the SDO_COORD_OP_PARAM_VALS table (described in [Section 6.6.5](#)).

Usage Notes

For information about using procedures to update well-known text (WKT) description, see [Section 6.7.1.2](#).

Examples

The following example updates the WKT description for all EPSG coordinate reference systems associated with coordinate transformation operation 2000067.

```
EXECUTE SDO_CS.UPDATE_WKTS_FOR_EPSG_OP(2000067);  
Updating SRID 20000671...
```

SDO_CS.UPDATE_WKTS_FOR_EPSG_PARAM

Format

```
SDO_CS.UPDATE_WKTS_FOR_EPSG_PARAM(  
    coord_op_id IN NUMBER,  
    parameter_id IN NUMBER);
```

Description

Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified coordinate transformation operation and parameter for transformation operations.

Parameters

coord_op_id

The ID of the SRID of the coordinate transformation operation. Must match a value in the COORD_OP_ID column of the SDO_COORD_OP_PARAM_VALS table (described in [Section 6.6.5](#)).

parameter_id

The ID of the SRID of the parameter for transformation operations. Must match a value in the PARAMETER_ID column of the SDO_COORD_OP_PARAM_VALS table (described in [Section 6.6.5](#)) where the COORD_OP_ID column value is equal to the coord_op_id parameter value.

Usage Notes

For information about using procedures to update well-known text (WKT) description, see [Section 6.7.1.2](#).

Examples

The following example updates the WKT description for all EPSG coordinate reference systems associated with coordinate transformation operation 9601 and parameter 8602.

```
EXECUTE SDO_CS.UPDATE_WKTS_FOR_EPSG_PARAM(9601, 8602);
```

SDO_CS.UPDATE_WKTS_FOR_EPSG_PM

Format

```
SDO_CS.UPDATE_WKTS_FOR_EPSG_PM(  
    prime_meridian_id IN NUMBER);
```

Description

Updates the well-known text (WKT) description for all EPSG coordinate reference systems associated with a specified prime meridian.

Parameters

prime_meridian_id

The ID of the prime meridian. Must match a value in the PRIME_MERIDIAN_ID column in the SDO_PRIME_MERIDIANS table (described in [Section 6.6.26](#)).

Usage Notes

For information about using procedures to update well-known text (WKT) description, see [Section 6.7.1.2](#).

Examples

The following example updates the WKT description for all EPSG coordinate reference systems associated with prime meridian 8902.

```
EXECUTE SDO_CS.UPDATE_WKTS_FOR_EPSG_PM(8902);  
Updating SRID 4803...  
Updating SRID 20790...  
Updating SRID 20791...  
Updating SRID 68036405...  
Updating SRID 4904...  
Updating SRID 2963...  
Updating SRID 69046405...
```

SDO_CS.VALIDATE_WKT

Format

```
SDO_CS.VALIDATE_WKT(
    srid IN NUMBER
) RETURN VARCHAR2;
```

Description

Validates the well-known text (WKT) description associated with a specified SRID.

Parameters

srid

The SRID of the coordinate system whose well-known text (WKT) description is to be validated. An entry for the specified value must exist in the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)).

Usage Notes

This function returns the string 'TRUE' if the WKT description is valid. If the WKT description is invalid, this function returns a string in the format 'FALSE (<position-number>)', where <position-number> is the number of the character position in the WKT description where the first error occurs.

The WKT description is checked to see if it satisfies the requirements described in [Section 6.7.1.1](#).

Examples

The following example validates the WKT description of the coordinate system associated with SRID 81989000. The results show that the cause of the invalidity (or the first cause of the invalidity) starts at character position 181 in the WKT description. (SRID 81989000 is not associated with any established coordinate system. Rather, it is for a deliberately invalid coordinate system that was inserted into a test version of the MDSYS.CS_SRS table, and it is not included in the MDSYS.CS_SRS table that is shipped with Oracle Spatial.)

```
SELECT SDO_CS.VALIDATE_WKT(81989000) FROM DUAL;
```

```
SDO_CS.VALIDATE_WKT(81989000)
```

```
-----
FALSE (181)
```

SDO_CS.VIEWPORT_TRANSFORM

Format

```
SDO_CS.VIEWPORT_TRANSFORM(  
    geom      IN SDO_GEOMETRY,  
    to_srid   IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

Description

Transforms an optimized rectangle into a valid polygon for use with Spatial operators and functions.

Note: This function is deprecated, and will not be supported in future releases of Spatial. Instead, use a geodetic MBR to specify the query window, as explained in [Section 6.2.3](#).

Parameters

geom

Geometry whose representation is to be transformed from an optimized rectangle to a valid polygon. The input geometry must have an SRID value of 0 (zero), as explained in the Usage Notes.

to_srid

The SRID of the coordinate system to be used for the transformation (that is, the SRID to be used in the returned geometry). `to_srid` must be either a value in the SRID column of the SDO_COORD_REF_SYS table (described in [Section 6.6.9](#)) or NULL.

Usage Notes

The geometry passed in must be an optimized rectangle.

If `to_srid` is a geodetic SRID, a geometry (not an optimized rectangle) is returned that conforms to the Oracle Spatial requirements for a geodetic geometry (for example, each polygon element's area must be less than one-half the surface area of the Earth).

If `to_srid` is not a geodetic SRID, an optimized rectangle is returned in which the SRID is set to `to_srid`.

Visualizer applications that work on geodetic data usually treat the longitude and latitude space as a regular Cartesian coordinate system. Fetching the data corresponding to a viewport is usually done with the help of an [SDO_FILTER](#) or [SDO_GEOM.RELATE](#) operation where the viewport (with an optimized rectangle representation) is sent as the window query. Before release 10.1, this optimized rectangle type could not be used in geodetic space, and therefore this type of viewport query could not be sent to the database. The VIEWPORT_TRANSFORM function was created to provide a workaround to this previous restriction.

The viewport rectangles should be constructed with the SRID value as 0 and input to the function to generate a corresponding valid geodetic polygon. This geodetic polygon can then be used in the [SDO_FILTER](#) or [SDO_GEOM.RELATE](#) call as the window object.

An SRID value of 0 should only be specified when calling the VIEWPORT_TRANSFORM function. It is not valid in any other context in Spatial.

This function should be used only when the display space is equirectangular (a rectangle), and the data displayed is geodetic.

Examples

The following example specifies the viewport as the whole Earth represented by an optimized rectangle. It returns the names of all four cola markets. (This example uses the definitions from the example in [Section 6.11](#).)

```
SELECT c.name FROM cola_markets_cs c WHERE
  SDO_FILTER(c.shape, SDO_CS.VIEWPORT_TRANSFORM(
    SDO_GEOMETRY(
      2003,
      0, -- SRID = 0 (special case)
      NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(-180,-90,180,90)),
    8307)) = 'TRUE';
```

NAME

```
-----
cola_a
cola_c
cola_b
cola_d
```

If the optimizer does not generate an optimal plan and performance is not as you expect, you can try the following alternative version of the query.

```
SELECT c.name FROM cola_markets_cs c,
  (SELECT
    SDO_CS.VIEWPORT_TRANSFORM(
      SDO_GEOMETRY(2003, 0, NULL,
        SDO_ELEM_INFO_ARRAY(1,1003,3),
        SDO_ORDINATE_ARRAY(-180,-90,180,90)), 8307)
    window_geom FROM DUAL)
  WHERE SDO_FILTER(c.shape, window_geom) = 'TRUE';
```

NAME

```
-----
cola_a
cola_c
cola_b
cola_d
```

SDO_GCDR Package (Geocoding)

The MDSYS.SDO_GCDR package contains subprograms for geocoding address data.

To use the subprograms in this chapter, you must understand the conceptual and usage information about geocoding in [Chapter 5](#).

[Table 14–1](#) lists the geocoding subprograms.

Table 14–1 Subprograms for Geocoding Address Data

Subprogram	Description
SDO_GCDR.GEOCODE	Geocodes an unformatted address and returns an SDO_GEO_ADDR object.
SDO_GCDR.GEOCODE_ADDR	Geocodes an input address using attributes in an SDO_GEO_ADDR object, and returns the first matched address as an SDO_GEO_ADDR object.
SDO_GCDR.GEOCODE_ADDR_ALL	Geocodes an input address using attributes in an SDO_GEO_ADDR object, and returns matching addresses as an SDO_ADDR_ARRAY object.
SDO_GCDR.GEOCODE_ALL	Geocodes all addresses associated with an unformatted address and returns the result as an SDO_ADDR_ARRAY object.
SDO_GCDR.GEOCODE_AS_GEOMETRY	Geocodes an unformatted address and returns an SDO_GEOMETRY object.
SDO_GCDR.REVERSE_GEOCODE	Reverse geocodes a location, specified by its spatial geometry object and country, and returns an SDO_GEO_ADDR object.

The rest of this chapter provides reference information on the subprograms, listed in alphabetical order.

SDO_GCDR.GEOCODE

Format

```
SDO_GCDR.GEOCODE(
    username    IN VARCHAR2,
    addr_lines   IN SDO_KEYWORDARRAY,
    country      IN VARCHAR2,
    match_mode  IN VARCHAR2
) RETURN SDO_GEO_ADDR;
```

Description

Geocodes an unformatted address and returns the result as an SDO_GEO_ADDR object.

Parameters

username

Name of the user that owns the tables containing the geocoding data.

addr_lines

An array of quoted strings representing the unformatted address to be geocoded. The SDO_KEYWORDARRAY type is described in [Section 5.2.3](#).

country

Country name or ISO country code.

match_mode

Match mode for the geocoding operation. Match modes are explained in [Section 5.1.2](#).

Usage Notes

This function returns an object of type SDO_GEO_ADDR, which is described in [Section 5.2.1](#). It performs the same operation as the [SDO_GCDR.GEOCODE_AS_GEOMETRY](#) function; however, that function returns an SDO_GEOMETRY object.

Examples

The following example geocodes the address of City Hall in San Francisco, California, using the RELAX_BASE_NAME match mode. It returns the longitude and latitude coordinates of this address as -122.41815 and 37.7784183, respectively.

```
SELECT SDO_GCDR.GEOCODE('SCOTT', SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl',
    'San Francisco, CA 94102'), 'US', 'RELAX_BASE_NAME') FROM DUAL;

SDO_GCDR.GEOCODE('SCOTT',SDO_KEYWORDARRAY('1CARLTONBGODLETTPL','SANFRANCISCO
-----
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), NULL, 'CARLTON B GOODLETT PL', NULL, NULL, '
SAN FRANCISCO', NULL, 'CA', 'US', '94102', NULL, '94102', NULL, '1', 'CARLTON B
GOODLETT', 'PL', 'F', 'F', NULL, NULL, 'L', .01, 23614360, 'nul?#ENUT?B281CP?',
1, 'DEFAULT', -122.41815, 37.7784183)
```

SDO_GCDR.GEOCODE_ADDR

Format

```
SDO_GCDR.GEOCODE_ADDR(
    gc_username IN VARCHAR2,
    address      IN SDO_GEO_ADDR
) RETURN SDO_GEO_ADDR;
```

Description

Geocodes an input address using attributes in an SDO_GEO_ADDR object, and returns the first matched address as an SDO_GEO_ADDR object.

Parameters

gc_username

Name of the user that owns the tables containing the geocoding data.

address

An SDO_GEO_ADDR object with one or more attributes set. The SDO_GEO_ADDR type is described in [Section 5.2.1](#).

Usage Notes

This function enables you to specify as many attributes in the input SDO_GEO_ADDR object as you can or want to set. It finds the first matching address, and returns an SDO_GEO_ADDR object with all possible attributes set.

Unlike the [SDO_GCDR.GEOCODE](#) function, which geocodes input addresses specified by unformatted address lines, the SDO_GCDR.GEOCODE_ADDR function input addresses specified by individual addressing fields defined in SDO_GEO_ADDR objects. When you use unformatted address lines, you rely on the geocoding software to parse the input address and decompose it into individual address fields. This process usually works well, but it can produce undesirable results if the input addresses are not well formatted. By contrast, when you specify parts of the input address as SDO_GEO_ADDR object attributes, you can reduce the chance of geocoding errors and produce more desirable results.

For examples of the SDO_GCDR.GEOCODE_ADDR function, see [Example 5-2](#) and [Example 5-3](#) in [Section 5.4](#).

See also the [SDO_GCDR.GEOCODE_ADDR_ALL](#) function, which performs the same operation as this function, but which can return more than one address.

Examples

The following example returns the geocoded result for a point of interest named CALIFORNIA PACIFIC MEDICAL CTR. The example uses a user-defined function named `create_addr_from_placename` (as defined in [Example 5-2](#) in [Section 5.4](#)) to construct the input SDO_GEO_ADDR object.

```
SELECT sdo_gcdr.geocode_addr('SCOTT',
    create_addr_from_placename('CALIFORNIA PACIFIC MEDICAL CTR', 'US'))
FROM DUAL;
```

```
SDO_GCDR.GEOCODE_ADDR('SCOTT',CREATE_ADDR_FROM_PLACENAME('CALIFORNIA PACIFIC ME
-----
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), NULL, 'BUCHANAN ST', NULL, NULL, 'SAN FRANCI
SCO', NULL, 'CA', 'US', '94115', NULL, '94115', NULL, '2333', NULL, NULL, 'F', '
F', NULL, NULL, 'L', 0, 23599031, '?????????B281CP?', 4, 'DEFAULT', -122.43097,
37.79138)
```

SDO_GCDR.GEOCODE_ADDR_ALL

Format

```
SDO_GCDR.GEOCODE_ADDR_ALL(
    gc_username IN VARCHAR2,
    address      IN SDO_GEO_ADDR,
    max_res_num IN NUMBER DEFAULT 4000
) RETURN SDO_ADDR_ARRAY;
```

Description

Geocodes an input address using attributes in an SDO_GEO_ADDR object, and returns matching addresses as an SDO_ADDR_ARRAY object (described in [Section 5.2.2](#)).

Parameters

gc_username

Name of the user that owns the tables containing the geocoding data.

address

An SDO_GEO_ADDR object with one or more attributes set. The SDO_GEO_ADDR type is described in [Section 5.2.1](#).

max_res_num

Maximum number of results to return in the SDO_ADDR_ARRAY object. The default value is 4000.

Usage Notes

This function enables you to specify as many attributes in the input SDO_GEO_ADDR object as you can or want to set. It finds matching addresses (up to 4000 or the limit specified in the `max_res_num` parameter), and returns an SDO_ADDR_ARRAY object in which each geocoded result has all possible attributes set.

This function performs the same operation as the [SDO_GCDR.GEOCODE_ADDR](#) function, except that it can return more than one address. See the Usage Notes for the [SDO_GCDR.GEOCODE_ADDR](#) function for more information.

Examples

The following example returns up to three geocoded results for a point of interest named CALIFORNIA PACIFIC MEDICAL CTR. (In this case only one result is returned, because the geocoding data contains only one address matching that point of interest.) The example uses a user-defined function named `create_addr_from_placename` (as defined in [Example 5-2](#) in [Section 5.4](#)) to construct the input SDO_GEO_ADDR object.

```
SELECT sdo_gcdr.geocode_addr_all('SCOTT',
    create_addr_from_placename('CALIFORNIA PACIFIC MEDICAL CTR', 'US'), 3)
FROM DUAL;
```

```
SDO_GCDR.GEOCODE_ADDR_ALL('SCOTT',CREATE_ADDR_FROM_PLACENAME('CALIFORNIAPACIF
-----
```

```
SDO_ADDR_ARRAY(SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), 'CALIFORNIA PACIFIC MEDICAL C  
TR-SF', 'BUCHANAN ST', NULL, NULL, 'SAN FRANCISCO', NULL, 'CA', 'US', '94115', N  
ULL, '94115', NULL, '2333', NULL, NULL, 'F', 'F', NULL, NULL, 'L', 0, 23599031,  
'?????????B281CP?', 4, 'DEFAULT', -122.43097, 37.79138))
```

SDO_GCDR.GEOCODE_ALL

Format

```
SDO_GCDR.GEOCODE_ALL(  
    gc_username IN VARCHAR2,  
    addr_lines   IN SDO_KEYWORDARRAY,  
    country      IN VARCHAR2,  
    match_mode   IN VARCHAR2  
) RETURN SDO_ADDR_ARRAY;
```

Description

Geocodes all addresses associated with an unformatted address and returns the result as an SDO_ADDR_ARRAY object.

Parameters

gc_username

Name of the user that owns the tables containing the geocoding data.

addr_lines

An array of quoted strings representing the unformatted address to be geocoded. The SDO_KEYWORDARRAY type is described in [Section 5.2.3](#).

country

Country name or ISO country code.

match_mode

Match mode for the geocoding operation. Match modes are explained in [Section 5.1.2](#).

Usage Notes

This function returns an object of type SDO_ADDR_ARRAY, which is described in [Section 5.2.2](#). It performs the same operation as the SDO_GCDR.GEOCODE function; however, it can return results for multiple addresses, in which case the returned SDO_ADDR_ARRAY object contains multiple SDO_GEO_ADDR objects. If your application needs to select one of the addresses for some further operations, you can use the information about each returned address to help you make that selection.

Each SDO_GEO_ADDR object in the returned SDO_ADDR_ARRAY array represents the center point of each street segment that matches the criteria in the addr_lines parameter. For example, if Main Street extends into two postal codes, or if there are two separate streets named Main Street in two separate postal codes, and if you specify Main Street and a city and state for this function, the returned SDO_ADDR_ARRAY array contains two SDO_GEO_ADDR objects, each reflecting the center point of Main Street in a particular postal code. The house or building number in each SDO_GEO_ADDR object is the house or building number located at the center point of the street segment, even if the input address contains no house or building number or a nonexistent number.

Examples

The following example returns an array of geocoded results, each result reflecting the center point of Clay Street in all postal codes in San Francisco, California, in which the street extends. The resulting array includes four SDO_GEO_ADDR objects, each reflecting the house at the center point of the Clay Street segment in each of the four postal codes (94108, 94115, 94118, and 94109) into which Clay Street extends.

```
SELECT SDO_GCDR.GEOCODE_ALL('SCOTT',
  SDO_KEYWORDARRAY('Clay St', 'San Francisco, CA'),
  'US', 'DEFAULT') FROM DUAL;

SDO_GCDR.GEOCODE_ALL('SCOTT',SDO_KEYWORDARRAY('CLAYST','SANFRANCISCO,CA'),'US
-----
SDO_ADDR_ARRAY(SDO_GEO_ADDR(1, SDO_KEYWORDARRAY(), NULL, 'CLAY ST', NULL, NULL,
'SAN FRANCISCO', NULL, 'CA', 'US', '94108', NULL, '94108', NULL, '978', 'CLAY',
'ST', 'F', 'F', NULL, NULL, 'L', 0, 23600689, 'nul?#ENUT?B281CP?', 1, 'DEFAULT',
-122.40904, 37.79385), SDO_GEO_ADDR(1, SDO_KEYWORDARRAY(), NULL, 'CLAY ST',
NULL, NULL, 'SAN FRANCISCO', NULL, 'CA', 'US', '94115', NULL, '94115', NULL, '27
98', 'CLAY', 'ST', 'F', 'F', NULL, NULL, 'L', 0, 23600709, 'nul?#ENUT?B281CP?',
1, 'DEFAULT', -122.43909, 37.79007), SDO_GEO_ADDR(1, SDO_KEYWORDARRAY(), NULL
, 'CLAY ST', NULL, NULL, 'SAN FRANCISCO', NULL, 'CA', 'US', '94118', NULL, '9411
8', NULL, '3698', 'CLAY', 'ST', 'F', 'F', NULL, NULL, 'L', 0, 23600718, 'nul?
#ENUT?B281CP?', 1, 'DEFAULT', -122.45372, 37.78822), SDO_GEO_ADDR(1, SDO_KEYWORD
ARRAY(), NULL, 'CLAY ST', NULL, NULL, 'SAN FRANCISCO', NULL, 'CA', 'US', '94109'
, NULL, '94109', NULL, '1698', 'CLAY', 'ST', 'F', 'F', NULL, NULL, 'L', 0, 23
600700, 'nul?#ENUT?B281CP?', 1, 'DEFAULT', -122.42093, 37.79236))
```

SDO_GCDR.GEOCODE_AS_GEOMETRY

Format

```
SDO_GCDR.GEOCODE_AS_GEOMETRY(
    username    IN VARCHAR2,
    addr_lines  IN SDO_KEYWORDARRAY,
    country     IN VARCHAR2
) RETURN SDO_GEOMETRY;
```

Description

Geocodes an unformatted address and returns the result as an SDO_GEOMETRY object.

Parameters

username

Name of the user that owns the tables containing the geocoding data.

addr_lines

An array of quoted strings representing the unformatted address to be geocoded. The SDO_KEYWORDARRAY type is described in [Section 5.2.3](#).

country

Country name or ISO country code.

Usage Notes

This function returns an object of type SDO_GEOMETRY. It performs the same operation as the [SDO_GCDR.GEOCODE](#) function; however, that function returns an SDO_GEOADDR object.

This function uses a match mode of 'DEFAULT' for the geocoding operation. Match modes are explained in [Section 5.1.2](#).

Examples

The following example geocodes the address of City Hall in San Francisco, California. It returns an SDO_GEOMETRY object in which the longitude and latitude coordinates of this address are -122.41815 and 37.7784183, respectively.

```
SELECT SDO_GCDR.GEOCODE_AS_GEOMETRY('SCOTT',
    SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA 94102'),
    'US') FROM DUAL;

SDO_GCDR.GEOCODE_AS_GEOMETRY('SCOTT',SDO_KEYWORDARRAY('1CARLTONBGODLETTPL',
-----
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.41815, 37.7784183, NULL), NULL, NUL
L)
```

SDO_GCDR.REVERSE_GEOCODE

Format

```
SDO_GCDR.REVERSE_GEOCODE(
  username IN VARCHAR2,
  location  IN SDO_GEOMETRY,
  country   IN VARCHAR2
) RETURN SDO_GEO_ADDR;
```

Description

Reverse geocodes a location, specified by its spatial geometry object and country, and returns the result as an SDO_GEO_ADDR object.

Parameters

username

Name of the user that owns the tables containing the geocoding data.

location

An SDO_GEOMETRY object that specifies the point location to be reverse geocoded.

country

Country name or ISO country code.

Usage Notes

This function returns an object of type SDO_GEO_ADDR, which is described in [Section 5.2.1](#).

A spatial index must be created on the table GC_ROAD_SEGMENT_<table-suffix>.

Examples

The following example reverse geocodes a point with the longitude and latitude values (-122.41815, 37.7784183). For this example, a spatial index was created on the GEOMETRY column in the GC_WORD_SEGMENT_US table.

```
SELECT SDO_GCDR.REVERSE_GEOCODE('SCOTT',
  SDO_GEOMETRY(2001, 8307,
    SDO_POINT_TYPE(-122.41815, 37.7784183, NULL), NULL, NULL),
  'US') FROM DUAL;
```

```
SDO_GCDR.REVERSE_GEOCODE('SCOTT',SDO_GEOMETRY(2001,8307,SDO_POINT_TYPE(-122.418
-----
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), NULL, 'POLK ST', NULL, NULL, 'SAN FRANCISCO'
, NULL, 'CA', 'US', '94102', NULL, '94102', NULL, '200', NULL, NULL, 'F', 'F', N
ULL, NULL, 'R', .00966633, 23614360, '', 1, 'DEFAULT', -122.41815, 37.7784177)
```

SDO_GEOM Package (Geometry)

This chapter contains descriptions of the geometry-related PL/SQL subprograms in the SDO_GEOM package, which can be grouped into the following categories:

- Relationship (True/False) between two objects: RELATE, WITHIN_DISTANCE
- Validation: VALIDATE_GEOMETRY_WITH_CONTEXT, VALIDATE_LAYER_WITH_CONTEXT
- Single-object operations: SDO_ARC_DENSIFY, SDO_AREA, SDO_BUFFER, SDO_CENTROID, SDO_CONVEXHULL, SDO_LENGTH, SDO_MAX_MBR_ORDINATE, SDO_MIN_MBR_ORDINATE, SDO_MBR, SDO_POINTONSURFACE
- Two-object operations: SDO_DISTANCE, SDO_DIFFERENCE, SDO_INTERSECTION, SDO_UNION, SDO_XOR

The geometry subprograms are listed [Table 15–1](#), and some usage information follows the table.

Table 15–1 *Geometry Subprograms*

Subprogram	Description
SDO_GEOM.RELATE	Determines how two objects interact.
SDO_GEOM.SDO_ARC_DENSIFY	Changes each circular arc into an approximation consisting of straight lines, and each circle into a polygon consisting of a series of straight lines that approximate the circle.
SDO_GEOM.SDO_AREA	Computes the area of a two-dimensional polygon.
SDO_GEOM.SDO_BUFFER	Generates a buffer polygon around or inside a geometry.
SDO_GEOM.SDO_CENTROID	Returns the centroid of a polygon.
SDO_GEOM.SDO_CONVEXHULL	Returns a polygon-type object that represents the convex hull of a geometry object.
SDO_GEOM.SDO_DIFFERENCE	Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects.
SDO_GEOM.SDO_DISTANCE	Computes the distance between two geometry objects.
SDO_GEOM.SDO_INTERSECTION	Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.

Table 15–1 (Cont.) Geometry Subprograms

Subprogram	Description
SDO_GEOM.SDO_LENGTH	Computes the length or perimeter of a geometry.
SDO_GEOM.SDO_MAX_MBR_ORDINATE	Returns the maximum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_MBR	Returns the minimum bounding rectangle of a geometry.
SDO_GEOM.SDO_MIN_MBR_ORDINATE	Returns the minimum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_POINTONSURFACE	Returns a point that is guaranteed to be on the surface of a polygon.
SDO_GEOM.SDO_UNION	Returns a geometry object that is the topological union (OR operation) of two geometry objects.
SDO_GEOM.SDO_XOR	Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects.
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT	Determines if a geometry is valid, and returns context information if the geometry is invalid.
SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT	Determines if all geometries stored in a column are valid, and returns context information about any invalid geometries.
SDO_GEOM.WITHIN_DISTANCE	Determines if two geometries are within a specified distance from one another.

The following usage information applies to the geometry subprograms. (See also the Usage Notes under the reference information for each subprogram.)

- Certain combinations of input parameters and operations can return a null value, that is, an empty geometry. For example, requesting the intersection of two disjoint geometry objects returns a null value.
- A null value (empty geometry) as an input parameter to a geometry function (for example, [SDO_GEOM.RELATE](#)) produces an error.
- Certain operations can return a geometry of a different type than one or both input geometries. For example, the intersection of a line and an overlapping polygon returns a line; the intersection of two lines returns a point; and the intersection of two tangent polygons returns a line.

SDO_GEOM.RELATE

Format

```
SDO_GEOM.RELATE(  
    geom1 IN SDO_GEOMETRY,  
    dim1  IN SDO_DIM_ARRAY,  
    mask  IN VARCHAR2,  
    geom2 IN SDO_GEOMETRY,  
    dim2  IN SDO_DIM_ARRAY  
    ) RETURN VARCHAR2;
```

or

```
SDO_GEOM.RELATE(  
    geom1 IN SDO_GEOMETRY,  
    mask  IN VARCHAR2,  
    geom2 IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    ) RETURN VARCHAR2;
```

Description

Examines two geometry objects to determine their spatial relationship.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

mask

Specifies a list of relationships to check. See the list of keywords in the Usage Notes.

geom2

Geometry object.

dim2

Dimensional information array corresponding to `geom2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

For better performance, use the [SDO_RELATE](#) operator or one of its convenience operator formats (all described in [Chapter 11](#)) instead of the `SDO_GEOM.RELATE`

function, unless you need to use the function. For example, the DETERMINE mask keyword does not apply with the [SDO_RELATE](#) operator. For more information about performance considerations with operators and functions, see [Section 1.9](#).

The `SDO_GEOM.RELATE` function can return the following types of answers:

- If you pass a `mask` listing one or more relationships, the function returns the name of the relationship if it is true for the pair of geometries. If all relationships are false, the procedure returns `FALSE`.
- If you pass the `DETERMINE` keyword in `mask`, the function returns the one relationship keyword that best matches the geometries.
- If you pass the `ANYINTERACT` keyword in `mask`, the function returns `TRUE` if the two geometries are not disjoint.

The following `mask` relationships can be tested:

- `ANYINTERACT`: Returns `TRUE` if the objects are not disjoint.
- `CONTAINS`: Returns `CONTAINS` if the second object is entirely within the first object and the object boundaries do not touch; otherwise, returns `FALSE`.
- `COVEREDBY`: Returns `COVEREDBY` if the first object is entirely within the second object and the object boundaries touch at one or more points; otherwise, returns `FALSE`.
- `COVERS`: Returns `COVERS` if the second object is entirely within the first object and the boundaries touch in one or more places; otherwise, returns `FALSE`.
- `DISJOINT`: Returns `DISJOINT` if the objects have no common boundary or interior points; otherwise, returns `FALSE`.
- `EQUAL`: Returns `EQUAL` if the objects share every point of their boundaries and interior, including any holes in the objects; otherwise, returns `FALSE`.
- `INSIDE`: Returns `INSIDE` if the first object is entirely within the second object and the object boundaries do not touch; otherwise, returns `FALSE`.
- `ON`: Returns `ON` if the boundary and interior of a line (the first object) is completely on the boundary of a polygon (the second object); otherwise, returns `FALSE`.
- `OVERLAPBDYDISJOINT`: Returns `OVERLAPBDYDISJOINT` if the objects overlap, but their boundaries do not interact; otherwise, returns `FALSE`.
- `OVERLAPBDYINTERSECT`: Returns `OVERLAPBDYINTERSECT` if the objects overlap, and their boundaries intersect in one or more places; otherwise, returns `FALSE`.
- `TOUCH`: Returns `TOUCH` if the two objects share a common boundary point, but no interior points; otherwise, returns `FALSE`.

Values for `mask` can be combined using the logical Boolean operator `OR`. For example, `'INSIDE + TOUCH'` returns `'INSIDE + TOUCH'` or `'FALSE'` depending on the outcome of the test.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example finds the relationship between each geometry in the SHAPE column and the cola_b geometry. (The example uses the definitions and data from [Section 2.1](#). The output is reformatted for readability.)

```
SELECT c.name,  
       SDO_GEOM.RELATE(c.shape, 'determine', c_b.shape, 0.005) relationship  
FROM cola_markets c, cola_markets c_b WHERE c_b.name = 'cola_b';
```

```
NAME      RELATIONSHIP  
-----  
cola_a    TOUCH  
cola_b    EQUAL  
cola_c    OVERLAPBDYINTERSECT  
cola_d    DISJOINT
```

Related Topics

- [SDO_RELATE](#) operator

SDO_GEOM.SDO_ARC_DENSIFY

Format

```
SDO_GEOM.SDO_ARC_DENSIFY(  
    geom IN SDO_GEOMETRY,  
    dim   IN SDO_DIM_ARRAY  
    params IN VARCHAR2  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_ARC_DENSIFY(  
    geom IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    params IN VARCHAR2  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry in which each circular arc in the input geometry is changed into an approximation of the circular arc consisting of straight lines, and each circle is changed into a polygon consisting of a series of straight lines that approximate the circle.

Parameters

geom

Geometry object.

dim

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

params

A quoted string containing an arc tolerance value and optionally a unit value. See the Usage Notes for an explanation of the format and meaning.

Usage Notes

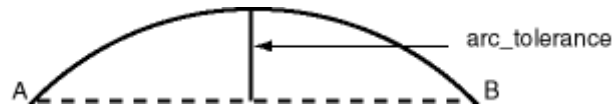
If you have geometries in a projected coordinate system that contain circles or circular arcs, you can use this function to densify them into regular polygons. You can then use the resulting straight-line polygon geometries for any Spatial operations, or you can transform them to any projected or geodetic coordinate system.

The `params` parameter is a quoted string that can contain the `arc_tolerance` keyword, as well as the `unit` keyword to identify the unit of measurement associated with the `arc_tolerance` value. For example:

```
'arc_tolerance=0.05 unit=km'
```

The `arc_tolerance` keyword specifies, for each arc in the geometry, the maximum length of the perpendicular line between the surface of the arc and the straight line between the start and end points of the arc. [Figure 15–1](#) shows a line whose length is the `arc_tolerance` value for the arc between points A and B.

Figure 15–1 Arc Tolerance



The `arc_tolerance` keyword value must be greater than the tolerance value associated with the geometry. (The default value for `arc_tolerance` is 20 times the tolerance value.) As you increase the `arc_tolerance` keyword value, the resulting polygon has fewer sides and a smaller area; as you decrease the `arc_tolerance` keyword value, the resulting polygon has more sides and a larger area (but never larger than the original geometry).

If the `unit` keyword is specified, the value must be an `SDO_UNIT` value from the `MDSYS.SDO_DIST_UNITS` table (for example, `'unit=KM'`). If the `unit` keyword is not specified, the unit of measurement associated with the geometry is used. See [Section 2.8](#) for more information about unit of measurement specification.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

Examples

The following example returns the geometry that results from the arc densification of `cola_d`, which is a circle. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Arc densification of the circle cola_d
SELECT c.name, SDO_GEOM.SDO_ARC_DENSIFY(c.shape, m.diminfo,
                                         'arc_tolerance=0.05')
FROM cola_markets c, user_sdo_geom_metadata m
WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
AND c.name = 'cola_d';

NAME
-----
SDO_GEOM.SDO_ARC_DENSIFY(C.SHAPE,M.DIMINFO,'ARC_TOLERANCE=0.05')(SDO_GTYPE, SDO_
-----
cola_d
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(8, 7, 8.76536686, 7.15224093, 9.41421356, 7.58578644, 9.84775907, 8.23463314,
 10, 9, 9.84775907, 9.76536686, 9.41421356, 10.4142136, 8.76536686, 10.8477591,
 8, 11, 7.23463314, 10.8477591, 6.58578644, 10.4142136, 6.15224093, 9.76536686, 6
, 9, 6.15224093, 8.23463314, 6.58578644, 7.58578644, 7.23463314, 7.15224093, 8,
 7))
```

Related Topics

- [Section 6.2.4, "Other Considerations and Requirements with Geodetic Data"](#)

SDO_GEOM.SDO_AREA

Format

```
SDO_GEOM.SDO_AREA(  
    geom IN SDO_GEOMETRY,  
    dim  IN SDO_DIM_ARRAY  
    [, unit IN VARCHAR2]  
    ) RETURN NUMBER;
```

or

```
SDO_GEOM.SDO_AREA(  
    geom IN SDO_GEOMETRY,  
    tol  IN NUMBER  
    [, unit IN VARCHAR2]  
    ) RETURN NUMBER;
```

Description

Returns the area of a two-dimensional polygon.

Parameters

geom

Geometry object.

dim

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

unit

Unit of measurement: a quoted string with `unit=` and an `SDO_UNIT` value from the `MDSYS.SDO_AREA_UNITS` table (for example, `'unit=SQ_KM'`). See [Section 2.8](#) for more information about unit of measurement specification.

If this parameter is not specified, the unit of measurement associated with the data is assumed. For geodetic data, the default unit of measurement is square meters.

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

This function works with any polygon, including polygons with holes.

Lines that close to form a ring have no area.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

Examples

The following example returns the areas of geometry objects stored in the COLA_MARKETS table. The first statement returns the areas of all objects; the second returns just the area of cola_a. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the areas of all cola markets.
```

```
SELECT name, SDO_GEOM.SDO_AREA(shape, 0.005) FROM cola_markets;
```

NAME	SDO_GEOM.SDO_AREA(SHAPE,0.005)
cola_a	24
cola_b	16.5
cola_c	5
cola_d	12.5663706

```
-- Return the area of just cola_a.
```

```
SELECT c.name, SDO_GEOM.SDO_AREA(c.shape, 0.005) FROM cola_markets c
WHERE c.name = 'cola_a';
```

NAME	SDO_GEOM.SDO_AREA(C.SHAPE,0.005)
cola_a	24

Related Topics

None.

SDO_GEOM.SDO_BUFFER

Format

```
SDO_GEOM.SDO_BUFFER(  
    geom    IN SDO_GEOMETRY,  
    dim     IN SDO_DIM_ARRAY,  
    dist    IN NUMBER  
    [, params IN VARCHAR2]  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_BUFFER(  
    geom    IN SDO_GEOMETRY,  
    dist    IN NUMBER,  
    tol     IN NUMBER  
    [, params IN VARCHAR2]  
    ) RETURN SDO_GEOMETRY;
```

Description

Generates a buffer polygon around or inside a geometry object.

Parameters

geom

Geometry object.

dim

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

dist

Distance value. If the value is positive, the buffer is generated around the geometry; if the value is negative (valid only for polygons), the buffer is generated inside the geometry. The absolute value of this parameter must be greater than the tolerance value, as specified in the dimensional array (`dim` parameter) or in the `tol` parameter.

tol

Tolerance value (see [Section 1.5.5](#)).

params

A quoted string that can contain one or both of the following keywords:

- `unit` and an `SDO_UNIT` value from the `MDSYS.SDO_DIST_UNITS` table. It identifies the unit of measurement associated with the `dist` parameter value, and also with the arc tolerance value if the `arc_tolerance` keyword is specified. See [Section 2.8](#) for more information about unit of measurement specification.

- `arc_tolerance` and an arc tolerance value. See the Usage Notes for the [SDO_GEOM.SDO_ARC_DENSIFY](#) function in this chapter for more information about the `arc_tolerance` keyword.

For example: `'unit=km arc_tolerance=0.05'`

If the input geometry is geodetic data and if `arc_tolerance` is not specified, the default value is the tolerance value multiplied by 20. Spatial uses the `arc_tolerance` value to perform arc densification in computing the result. If the input geometry is Cartesian or projected data, `arc_tolerance` has no effect and should not be specified.

If this parameter is not specified for a Cartesian or projected geometry, or if the `arc_tolerance` keyword is specified for a geodetic geometry but the `unit` keyword is not specified, the unit of measurement associated with the data is assumed.

Usage Notes

This function returns a geometry object representing the buffer polygon.

This function creates a rounded buffer around a point, line, or polygon, or inside a polygon. The buffer within a void is also rounded, and is the same distance from the inner boundary as the outer buffer is from the outer boundary. See [Figure 1-7](#) for an illustration.

If the buffer polygon geometry is in a projected coordinate system, it will contain arcs; and if you want to transform that geometry to a geodetic coordinate system, you must first densify it using the [SDO_GEOM.SDO_ARC_DENSIFY](#) function, and then transform the densified geometry.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

With geodetic data, this function is supported by approximations, as explained in [Section 6.9.3](#).

Examples

The following example returns a polygon representing a buffer of 1 around `cola_a`. Note the rounded corners (for example, at `.292893219,.292893219`) in the returned polygon. (The example uses the non-geodetic definitions and data from [Section 2.1](#).)

```
-- Generate a buffer of 1 unit around a geometry.
SELECT c.name, SDO_GEOM.SDO_BUFFER(c.shape, m.diminfo, 1)
   FROM cola_markets c, user_sdo_geom_metadata m
   WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
   AND c.name = 'cola_a';

NAME
-----
SDO_GEOM.SDO_BUFFER(C.SHAPE,M.DIMINFO,1)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z)
-----
cola_a
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1005, 8, 1, 2, 2, 5, 2, 1,
7, 2, 2, 11, 2, 1, 13, 2, 2, 17, 2, 1, 19, 2, 2, 23, 2, 1), SDO_ORDINATE_ARRAY(
0, 1, .292893219, .292893219, 1, 0, 5, 0, 5.70710678, .292893219, 6, 1, 6, 7, 5.
70710678, 7.70710678, 5, 8, 1, 8, .292893219, 7.70710678, 0, 7, 0, 1))
```

The following example returns a polygon representing a buffer of 1 around `cola_a` using the geodetic definitions and data from [Section 6.11](#).

```
-- Generate a buffer of 1 kilometer around a geometry.
```

```
SELECT c.name, SDO_GEOM.SDO_BUFFER(c.shape, m.diminfo, 1,
                                   'unit=km arc_tolerance=0.05')
FROM cola_markets c, user_sdo_geom_metadata m
WHERE m.table_name = 'COLA_MARKETS'
AND m.column_name = 'SHAPE' AND c.name = 'cola_a';

NAME
-----
SDO_GEOM.SDO_BUFFER(C.SHAPE,M.DIMINFO,1,'UNIT=KMARC_TOLERANCE=0.05')(SDO_GTYPE,
-----
cola_a
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(.991023822, 1.00002073, .992223711, .995486419, .99551726, .99217077, 1.00001
929, .990964898, 4.99998067, .990964929, 5.00448268, .9921708, 5.00777624, .9954
86449, 5.00897618, 1.00002076, 5.00904194, 6.99997941, 5.00784065, 7.00450033, 5
.00454112, 7.00781357, 5.00002479, 7.009034, .999975166, 7.00903403, .995458814,
7.00781359, .992159303, 7.00450036, .990958058, 6.99997944, .991023822, 1.00002
073))
```

Related Topics

- [SDO_GEOM.SDO_UNION](#)
- [SDO_GEOM.SDO_INTERSECTION](#)
- [SDO_GEOM.SDO_XOR](#)

SDO_GEOM.SDO_CENTROID

Format

```
SDO_GEOM.SDO_CENTROID(  
    geom1 IN SDO_GEOMETRY,  
    dim1  IN SDO_DIM_ARRAY  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_CENTROID(  
    geom1 IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns a point geometry that is the centroid of a polygon, multipolygon, point, or point cluster. (The centroid is also known as the "center of gravity.")

For an input geometry consisting of multiple objects, the result is weighted by the area of each polygon in the geometry objects. If the geometry objects are a mixture of polygons and points, the points are not used in the calculation of the centroid. If the geometry objects are all points, the points have equal weight.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

The function returns a null value if `geom1` is not a polygon, multipolygon, point, or point cluster, as identified by the `SDO_GTYPE` value in the `SDO_GEOMETRY` object.

If `geom1` is a point, the function returns the point (the input geometry).

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

With geodetic data, this function is supported by approximations, as explained in [Section 6.9.3](#).

Depending on the shape and complexity of the input geometry, the returned point might not be on the surface of the input geometry.

Examples

The following example returns a geometry object that is the centroid of `cola_c`. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the centroid of a geometry.
SELECT c.name, SDO_GEOM.SDO_CENTROID(c.shape, m.diminfo)
   FROM cola_markets c, user_sdo_geom_metadata m
   WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
   AND c.name = 'cola_c';

NAME
-----
SDO_GEOM.SDO_CENTROID(C.SHAPE,M.DIMINFO)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z)
-----
cola_c
SDO_GEOMETRY(2001, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
4.73333333, 3.93333333))
```

Related Topics

None.

SDO_GEOM.SDO_CONVEXHULL

Format

```
SDO_GEOM.SDO_CONVEXHULL(
    geom1 IN SDO_GEOMETRY,
    dim1  IN SDO_DIM_ARRAY
) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_CONVEXHULL(
    geom1 IN SDO_GEOMETRY,
    tol   IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns a polygon-type object that represents the convex hull of a geometry object.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

The **convex hull** is a simple convex polygon that completely encloses the geometry object. Spatial uses as few straight-line sides as possible to create the smallest polygon that completely encloses the specified object. A convex hull is a convenient way to get an approximation of a complex geometry object.

If the geometry (`geom1`) contains any arc elements, the function calculates the minimum bounding rectangle (MBR) for each arc element and uses these MBRs in calculating the convex hull of the geometry. If the geometry object (`geom1`) is a circle, the function returns a square that minimally encloses the circle.

The function returns a null value if `geom1` is of point type, has fewer than three points or vertices, or consists of multiple points all in a straight line.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

With geodetic data, this function is supported by approximations, as explained in [Section 6.9.3](#).

Examples

The following example returns a geometry object that is the convex hull of `cola_c`. (The example uses the definitions and data from [Section 2.1](#). This specific example, however, does not produce useful output—the returned polygon has the same vertices as the input polygon—because the input polygon is already a simple convex polygon.)

```
-- Return the convex hull of a polygon.
SELECT c.name, SDO_GEOM.SDO_CONVEXHULL(c.shape, m.diminfo)
   FROM cola_markets c, user_sdo_geom_metadata m
   WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
   AND c.name = 'cola_c';

NAME
-----
SDO_GEOM.SDO_CONVEXHULL(C.SHAPE,M.DIMINFO) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y,
-----
cola_c
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(6, 3, 6, 5, 4, 5, 3, 3, 6, 3))
```

Related Topics

None.

SDO_GEOM.SDO_DIFFERENCE

Format

```
SDO_GEOM.SDO_DIFFERENCE(  
    geom1 IN SDO_GEOMETRY,  
    dim1  IN SDO_DIM_ARRAY,  
    geom2 IN SDO_GEOMETRY,  
    dim2  IN SDO_DIM_ARRAY  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_DIFFERENCE(  
    geom1 IN SDO_GEOMETRY,  
    geom2 IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom2

Geometry object.

dim2

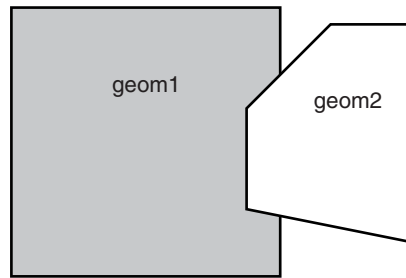
Dimensional information array corresponding to `geom2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

In [Figure 15-2](#), the shaded area represents the polygon returned when `SDO_DIFFERENCE` is used with a square (`geom1`) and another polygon (`geom2`).

Figure 15–2 SDO_GEOM.SDO_DIFFERENCE

If the function format with `tol` is used, all geometry objects must be defined using 4-digit SDO_GTYPE values (explained in [Section 2.2.1](#)).

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example returns a geometry object that is the topological difference (MINUS operation) of `cola_a` and `cola_c`. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the topological difference of two geometries.
SELECT SDO_GEOM.SDO_DIFFERENCE(c_a.shape, m.diminfo, c_c.shape, m.diminfo)
  FROM cola_markets c_a, cola_markets c_c, user_sdo_geom_metadata m
  WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
  AND c_a.name = 'cola_a' AND c_c.name = 'cola_c';

SDO_GEOM.SDO_DIFFERENCE(C_A.SHAPE,M.DIMINFO,C_C.SHAPE,M.DIMINFO) (SDO_GTYPE, SDO_
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(1, 7, 1, 1, 5, 1, 5, 3, 3, 3, 4, 5, 5, 5, 5, 7, 1, 7)
```

Note that in the returned polygon, the SDO_ORDINATE_ARRAY starts and ends at the same point (1, 7).

Related Topics

- [SDO_GEOM.SDO_INTERSECTION](#)
- [SDO_GEOM.SDO_UNION](#)
- [SDO_GEOM.SDO_XOR](#)

SDO_GEOM.SDO_DISTANCE

Format

```
SDO_GEOM.SDO_DISTANCE(
    geom1 IN SDO_GEOMETRY,
    dim1  IN SDO_DIM_ARRAY,
    geom2 IN SDO_GEOMETRY,
    dim2  IN SDO_DIM_ARRAY
    [, unit IN VARCHAR2]
) RETURN NUMBER;
```

or

```
SDO_GEOM.SDO_DISTANCE(
    geom1 IN SDO_GEOMETRY,
    geom2 IN SDO_GEOMETRY,
    tol   IN NUMBER
    [, unit IN VARCHAR2]
) RETURN NUMBER;
```

Description

Computes the distance between two geometry objects. The distance between two geometry objects is the distance between the closest pair of points or segments of the two objects.

Parameters

geom1

Geometry object whose distance from `geom2` is to be computed.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom2

Geometry object whose distance from `geom1` is to be computed.

dim2

Dimensional information array corresponding to `geom2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

unit

Unit of measurement: a quoted string with `unit=` and an `SDO_UNIT` value from the `MDSYS.SDO_DIST_UNITS` table (for example, `'unit=KM'`). See [Section 2.8](#) for more information about unit of measurement specification.

If this parameter is not specified, the unit of measurement associated with the data is assumed.

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example returns the shortest distance between `cola_b` and `cola_d`. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the distance between two geometries.
SELECT SDO_GEOM.SDO_DISTANCE(c_b.shape, c_d.shape, 0.005)
       FROM cola_markets c_b, cola_markets c_d
       WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';

SDO_GEOM.SDO_DISTANCE(C_B.SHAPE,C_D.SHAPE,0.005)
-----
                               .846049894
```

Related Topics

- [SDO_GEOM.WITHIN_DISTANCE](#)

SDO_GEOM.SDO_INTERSECTION

Format

```
SDO_GEOM.SDO_INTERSECTION(  
    geom1 IN SDO_GEOMETRY,  
    dim1  IN SDO_DIM_ARRAY,  
    geom2 IN SDO_GEOMETRY,  
    dim2  IN SDO_DIM_ARRAY  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_INTERSECTION(  
    geom1 IN SDO_GEOMETRY,  
    geom2 IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom2

Geometry object.

dim2

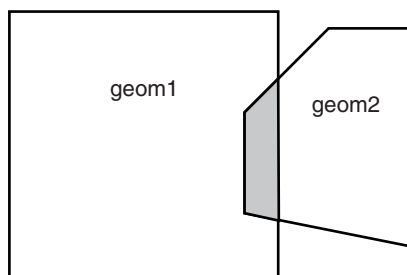
Dimensional information array corresponding to `geom2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

In [Figure 15-3](#), the shaded area represents the polygon returned when `SDO_INTERSECTION` is used with a square (`geom1`) and another polygon (`geom2`).

Figure 15–3 SDO_GEOM.SDO_INTERSECTION

If the function format with `tol` is used, all geometry objects must be defined using 4-digit SDO_GTYPE values (explained in [Section 2.2.1](#)).

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example returns a geometry object that is the topological intersection (AND operation) of `cola_a` and `cola_c`. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the topological intersection of two geometries.
SELECT SDO_GEOM.SDO_INTERSECTION(c_a.shape, c_c.shape, 0.005)
   FROM cola_markets c_a, cola_markets c_c
   WHERE c_a.name = 'cola_a' AND c_c.name = 'cola_c';

SDO_GEOM.SDO_INTERSECTION(C_A.SHAPE,C_C.SHAPE,0.005)(SDO_GTYPE, SDO_SRID, SDO_PO
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(4, 5, 3, 3, 5, 3, 5, 5, 4, 5))
```

Note that in the returned polygon, the SDO_ORDINATE_ARRAY starts and ends at the same point (4, 5).

Related Topics

- [SDO_GEOM.SDO_DIFFERENCE](#)
- [SDO_GEOM.SDO_UNION](#)
- [SDO_GEOM.SDO_XOR](#)

SDO_GEOM.SDO_LENGTH

Format

```
SDO_GEOM.SDO_LENGTH(
    geom IN SDO_GEOMETRY,
    dim  IN SDO_DIM_ARRAY
    [, unit IN VARCHAR2]
) RETURN NUMBER;
```

or

```
SDO_GEOM.SDO_LENGTH(
    geom IN SDO_GEOMETRY,
    tol  IN NUMBER
    [, unit IN VARCHAR2]
) RETURN NUMBER;
```

Description

Returns the length or perimeter of a geometry object.

Parameters

geom

Geometry object.

dim

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

unit

Unit of measurement: a quoted string with `unit=` and an `SDO_UNIT` value from the `MDSYS.SDO_DIST_UNITS` table (for example, 'unit=KM'). See [Section 2.8](#) for more information about unit of measurement specification.

If this parameter is not specified, the unit of measurement associated with the data is assumed. For geodetic data, the default unit of measurement is meters.

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

If the input polygon contains one or more holes, this function calculates the perimeters of the exterior boundary and all holes. It returns the sum of all perimeters.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

Examples

The following example returns the perimeters of geometry objects stored in the `COLA_MARKETS` table. The first statement returns the perimeters of all objects; the

second returns just the perimeter of cola_a. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the perimeters of all cola markets.
SELECT c.name, SDO_GEOM.SDO_LENGTH(c.shape, m.diminfo)
   FROM cola_markets c, user_sdo_geom_metadata m
   WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE';
```

NAME	SDO_GEOM.SDO_LENGTH(C.SHAPE,M.DIMINFO)
cola_a	20
cola_b	17.1622777
cola_c	9.23606798
cola_d	12.5663706

```
-- Return the perimeter of just cola_a.
SELECT c.name, SDO_GEOM.SDO_LENGTH(c.shape, m.diminfo)
   FROM cola_markets c, user_sdo_geom_metadata m
   WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
   AND c.name = 'cola_a';
```

NAME	SDO_GEOM.SDO_LENGTH(C.SHAPE,M.DIMINFO)
cola_a	20

Related Topics

None.

SDO_GEOM.SDO_MAX_MBR_ORDINATE

Format

```
SDO_GEOM.SDO_MAX_MBR_ORDINATE(
    geom          IN SDO_GEOMETRY,
    ordinate_pos  IN NUMBER
) RETURN NUMBER;
```

or

```
SDO_GEOM.SDO_MAX_MBR_ORDINATE(
    geom          IN SDO_GEOMETRY,
    dim           IN SDO_DIM_ARRAY,
    ordinate_pos  IN NUMBER
) RETURN NUMBER;
```

Description

Returns the maximum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.

Parameters

geom

Geometry object.

dim

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

ordinate_pos

Position of the ordinate (dimension) in the definition of the geometry object: 1 for the first ordinate, 2 for the second ordinate, and so on. For example, if `geom` has X, Y ordinates, 1 identifies the X ordinate and 2 identifies the Y ordinate.

Usage Notes

None.

Examples

The following example returns the maximum X (first) ordinate value of the minimum bounding rectangle of the `cola_d` geometry in the `COLA_MARKETS` table. (The example uses the definitions and data from [Section 2.1](#). The minimum bounding rectangle of `cola_d` is returned in the example for the [SDO_GEOM.SDO_MBR](#) function.)

```
SELECT SDO_GEOM.SDO_MAX_MBR_ORDINATE(c.shape, m.diminfo, 1)
       FROM cola_markets c, user_sdo_geom_metadata m
       WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
              AND c.name = 'cola_d';
```

```
SDO_GEOM.SDO_MAX_MBR_ORDINATE(C.SHAPE,M.DIMINFO,1)
```

Related Topics

- [SDO_GEOM.SDO_MBR](#)
- [SDO_GEOM.SDO_MIN_MBR_ORDINATE](#)

SDO_GEOM.SDO_MBR

Format

```
SDO_GEOM.SDO_MBR(
    geom IN SDO_GEOMETRY
    [, dim IN SDO_DIM_ARRAY]
) RETURN SDO_GEOMETRY;
```

Description

Returns the minimum bounding rectangle of a geometry object, that is, a single rectangle that minimally encloses the geometry.

Parameters

geom

Geometry object.

dim

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function does not return an MBR geometry if a proper MBR cannot be constructed. Specifically:

- If the input geometry is null, the function returns a null geometry.
- If the input geometry is a point, the function returns the point.
- If the input geometry consists of points all on a straight line, the function returns a two-point line.
- If the input geometry has three dimensions but all Z dimension values are the same, the function returns a three-dimensional line.

Examples

The following example returns the minimum bounding rectangle of the `cola_d` geometry in the `COLA_MARKETS` table. (The example uses the definitions and data from [Section 2.1](#). Because `cola_d` is a circle, the minimum bounding rectangle in this case is a square.)

```
-- Return the minimum bounding rectangle of cola_d (a circle).
SELECT SDO_GEOM.SDO_MBR(c.shape, m.diminfo)
FROM cola_markets c, user_sdo_geom_metadata m
WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
AND c.name = 'cola_d';
```

```
SDO_GEOM.SDO_MBR(C.SHAPE,M.DIMINFO)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(6, 7, 10, 11))
```

Related Topics

- [SDO_GEOM.SDO_MAX_MBR_ORDINATE](#)
- [SDO_GEOM.SDO_MIN_MBR_ORDINATE](#)

SDO_GEOM.SDO_MIN_MBR_ORDINATE

Format

```
SDO_GEOM.SDO_MIN_MBR_ORDINATE(
    geom          IN SDO_GEOMETRY,
    ordinate_pos  IN NUMBER
) RETURN NUMBER;
```

or

```
SDO_GEOM.SDO_MIN_MBR_ORDINATE(
    geom          IN SDO_GEOMETRY,
    dim           IN SDO_DIM_ARRAY,
    ordinate_pos  IN NUMBER
) RETURN NUMBER;
```

Description

Returns the minimum value for the specified ordinate (dimension) of the minimum bounding rectangle of a geometry object.

Parameters

geom

Geometry object.

dim

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

ordinate_pos

Position of the ordinate (dimension) in the definition of the geometry object: 1 for the first ordinate, 2 for the second ordinate, and so on. For example, if `geom` has X, Y ordinates, 1 identifies the X ordinate and 2 identifies the Y ordinate.

Usage Notes

None.

Examples

The following example returns the minimum X (first) ordinate value of the minimum bounding rectangle of the `cola_d` geometry in the `COLA_MARKETS` table. (The example uses the definitions and data from [Section 2.1](#). The minimum bounding rectangle of `cola_d` is returned in the example for the [SDO_GEOM.SDO_MBR](#) function.)

```
SELECT SDO_GEOM.SDO_MIN_MBR_ORDINATE(c.shape, m.diminfo, 1)
       FROM cola_markets c, user_sdo_geom_metadata m
       WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
              AND c.name = 'cola_d';
```

```
SDO_GEOM.SDO_MIN_MBR_ORDINATE(C.SHAPE,M.DIMINFO,1)
```

Related Topics

- [SDO_GEOM.SDO_MAX_MBR_ORDINATE](#)
- [SDO_GEOM.SDO_MBR](#)

SDO_GEOM.SDO_POINTONSURFACE

Format

```
SDO_GEOM.SDO_POINTONSURFACE(
    geom1 IN SDO_GEOMETRY,
    dim1  IN SDO_DIM_ARRAY
) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_POINTONSURFACE(
    geom1 IN SDO_GEOMETRY,
    tol   IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns a point that is guaranteed to be on the surface of a polygon geometry object.

Parameters

geom1

Polygon geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

This function returns a point geometry object representing a point that is guaranteed to be on the surface of `geom1`.

The returned point can be any point on the surface. You should not make any assumptions about where on the surface the returned point is, or about whether the point is the same or different when the function is called multiple times with the same input parameter values.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

Examples

The following example returns a geometry object that is a point on the surface of `cola_a`. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return a point on the surface of a geometry.
SELECT SDO_GEOM.SDO_POINTONSURFACE(c.shape, m.diminfo)
FROM cola_markets c, user_sdo_geom_metadata m
WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
AND c.name = 'cola_a';
```

```
SDO_GEOM.SDO_POINTONSURFACE(C.SHAPE,M.DIMINFO)(SDO_GTYPE, SDO_SRID, SDO_POINT(X,  
-----  
SDO_GEOMETRY(2001, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(  
1, 1))
```

Related Topics

None.

SDO_GEOM.SDO_UNION

Format

```
SDO_GEOM.SDO_UNION(  
    geom1 IN SDO_GEOMETRY,  
    dim1  IN SDO_DIM_ARRAY,  
    geom2 IN SDO_GEOMETRY,  
    dim2  IN SDO_DIM_ARRAY  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_UNION(  
    geom1 IN SDO_GEOMETRY,  
    geom2 IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry object that is the topological union (OR operation) of two geometry objects.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom2

Geometry object.

dim2

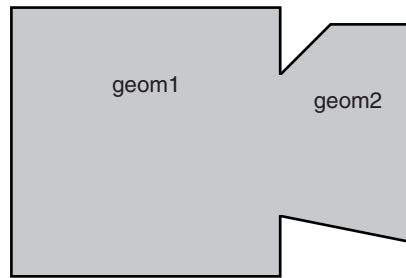
Dimensional information array corresponding to `geom2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

In [Figure 15-4](#), the shaded area represents the polygon returned when `SDO_UNION` is used with a square (`geom1`) and another polygon (`geom2`).

Figure 15–4 SDO_GEOM.SDO_UNION

If the function format with `tol` is used, all geometry objects must be defined using 4-digit SDO_GTYPE values (explained in [Section 2.2.1](#)).

If it is sufficient to append one geometry to another geometry without performing a topological union operation, and if both geometries are disjoint, using the [SDO_UTIL.APPEND](#) function (described in [Chapter 20](#)) is faster than using the SDO_UNION function.

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example returns a geometry object that is the topological union (OR operation) of `cola_a` and `cola_c`. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the topological union of two geometries.
SELECT SDO_GEOM.SDO_UNION(c_a.shape, m.diminfo, c_c.shape, m.diminfo)
  FROM cola_markets c_a, cola_markets c_c, user_sdo_geom_metadata m
  WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
  AND c_a.name = 'cola_a' AND c_c.name = 'cola_c';

SDO_GEOM.SDO_UNION(C_A.SHAPE,M.DIMINFO,C_C.SHAPE,M.DIMINFO) (SDO_GTYPE, SDO_SRID,
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(5, 5, 5, 7, 1, 7, 1, 1, 5, 1, 5, 3, 6, 3, 6, 5, 5, 5))
```

Note that in the returned polygon, the SDO_ORDINATE_ARRAY starts and ends at the same point (5, 5).

Related Topics

- [SDO_GEOM.SDO_DIFFERENCE](#)
- [SDO_GEOM.SDO_INTERSECTION](#)
- [SDO_GEOM.SDO_XOR](#)

SDO_GEOM.SDO_XOR

Format

```
SDO_GEOM.SDO_XOR(  
    geom1 IN SDO_XOR,  
    dim1  IN SDO_DIM_ARRAY,  
    geom2 IN SDO_GEOMETRY,  
    dim2  IN SDO_DIM_ARRAY  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_GEOM.SDO_XOR(  
    geom1 IN SDO_GEOMETRY,  
    geom2 IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom2

Geometry object.

dim2

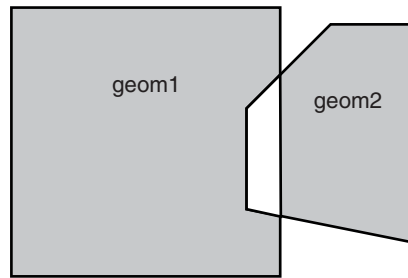
Dimensional information array corresponding to `geom2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

In [Figure 15-5](#), the shaded area represents the polygon returned when `SDO_XOR` is used with a square (`geom1`) and another polygon (`geom2`).

Figure 15–5 SDO_GEOM.SDO_XOR

If the function format with `tol` is used, all geometry objects must be defined using 4-digit SDO_GTYPE values (explained in [Section 2.2.1](#)).

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example returns a geometry object that is the topological symmetric difference (XOR operation) of `cola_a` and `cola_c`. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Return the topological symmetric difference of two geometries.
SELECT SDO_GEOM.SDO_XOR(c_a.shape, m.diminfo, c_c.shape, m.diminfo)
  FROM cola_markets c_a, cola_markets c_c, user_sdo_geom_metadata m
 WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
 AND c_a.name = 'cola_a' AND c_c.name = 'cola_c';

SDO_GEOM.SDO_XOR(C_A.SHAPE,M.DIMINFO,C_C.SHAPE,M.DIMINFO) (SDO_GTYPE, SDO_SRID, S
-----
SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1, 19, 1003, 1), SDO
_ORDINATE_ARRAY(1, 7, 1, 1, 5, 1, 5, 3, 3, 3, 4, 5, 5, 5, 5, 7, 1, 7, 5, 5, 5, 3
, 6, 3, 6, 5, 5, 5))
```

Note that the returned polygon is a multipolygon (SDO_GTYPE = 2007), and the SDO_ORDINATE_ARRAY describes two polygons: one starting and ending at (1, 7) and the other starting and ending at (5, 5).

Related Topics

- [SDO_GEOM.SDO_DIFFERENCE](#)
- [SDO_GEOM.SDO_INTERSECTION](#)
- [SDO_GEOM.SDO_UNION](#)

SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT

Format

```
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(  
    theGeometry IN SDO_GEOMETRY,  
    theDimInfo  IN SDO_DIM_ARRAY  
    ) RETURN VARCHAR2;
```

or

```
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(  
    theGeometry IN SDO_GEOMETRY,  
    tolerance   IN NUMBER  
    ) RETURN VARCHAR2;
```

Description

Performs a consistency check for valid geometry types and returns context information if the geometry is invalid. The function checks the representation of the geometry from the tables against the element definitions.

Parameters

theGeometry

Geometry object.

theDimInfo

Dimensional information array corresponding to `theGeometry`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tolerance

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

If the geometry is valid, this function returns TRUE. (For a user-defined geometry, that is, a geometry with an `SDO_GTYPE` value of 2000, this function returns the string NULL.)

If the geometry is not valid, this function returns the following:

- An Oracle error message number based on the specific reason the geometry is invalid, or FALSE if the geometry fails for some other reason
- The context of the error (the coordinate, edge, or ring that causes the geometry to be invalid). (See "[Context of Errors: Details](#)" in this section.)

This function checks for type consistency and geometry consistency.

For type consistency, the function checks for the following:

- The `SDO_GTYPE` is valid.

- The SDO_ETYPE values are consistent with the SDO_GTYPE value. For example, if the SDO_GTYPE is 2003, there should be at least one element of type POLYGON in the geometry.
- The SDO_ELEM_INFO_ARRAY has valid triplet values.

For geometry consistency, the function checks for the following, as appropriate for the specific geometry type:

- Polygons have at least four points, which includes the point that closes the polygon. (The last point is the same as the first.)
- Polygons are not self-crossing.
- No two vertices on a line or polygon are the same.
- Polygons are oriented correctly. (Exterior ring boundaries must be oriented counterclockwise, and interior ring boundaries must be oriented clockwise.)
- An interior polygon ring touches the exterior polygon ring at no more than one point.
- If two or more interior polygon rings are in an exterior polygon ring, the interior polygon rings touch at no more than one point.
- Line strings have at least two points.
- SDO_ETYPE 1-digit and 4-digit values are not mixed (that is, both used) in defining polygon ring elements.
- Points on an arc are not colinear (that is, are not on a straight line) and are not the same point.
- Geometries are within the specified bounds of the applicable DIMINFO column value (from the USER_SDO_GEOM_METADATA view).
- LRS geometries (see [Chapter 7](#)) have three or four dimensions and a valid measure dimension position (3 or 4, depending on the number of dimensions).

In checking for geometry consistency, the function considers the geometry's tolerance value in determining if lines touch or if points are the same.

If the function format with `tolerance` is used, the following guidelines apply:

- All geometry objects must be defined using 4-digit SDO_GTYPE values (explained in [Section 2.2.1](#)).
- No checking is done to validate that the geometry is within the coordinate system bounds as stored in the DIMINFO field of the USER_SDO_GEOM_METADATA view. If this check is required for your usage, use the function format with `theDimInfo`.

You can use this function in a PL/SQL procedure as an alternative to using the [SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT](#) procedure. See the Usage Notes for [SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT](#) for more information.

Context of Errors: Details

If a geometry is invalid, the result can include information about a combination of the following: coordinates, elements, rings, and edges.

- **Coordinates:** A coordinate refers to a vertex in a geometry. In a two-dimensional geometry, a vertex is two numbers (X and Y, or Longitude and Latitude). In a three-dimensional geometry, a vertex is defined using three numbers; and in a

four-dimensional geometry, a vertex is defined using four numbers. (You can use the [SDO_UTIL.GETVERTICES](#) function to return the coordinates in a geometry.)

If you receive a geometry validation error such as 13356 (adjacent points in a geometry are redundant), you can call the [SDO_UTIL.GETVERTICES](#) function, specifying a rownum stopping condition to include the coordinate one greater than the coordinate indicated with the error. The last two coordinates shown in the output are the redundant coordinates. These coordinates may be exactly the same, or they may be within the user-specified tolerance and thus are considered the same point. You can remove redundant coordinates by using the [SDO_UTIL.REMOVE_DUPLICATE_VERTICES](#) function.

- **Elements:** An element is a point, a line string, or an exterior polygon with zero or more corresponding interior polygons. (That is, a polygon element includes the exterior ring and all interior rings associated with that exterior ring.) If a geometry is a multi-element geometry (for example, multiple points, lines, or polygons), the first element is element 1, the second element is element 2, and so on.
- **Rings:** A ring is only used with polygon elements. Exterior rings in a polygon are considered polygon elements, and an exterior ring can include zero or more interior rings (or holes). Each interior ring has its own ring designation, but Ring 1 is associated with the exterior polygon itself. For example, Element 1, Ring 1 refers to the first exterior polygon in a geometry; Element 1, Ring 2 refers to the first interior polygon of the first exterior polygon; and Element 1, Ring 3 refers to the second interior polygon. If the geometry is a multipolygon, Element 2, Ring 1 is used to refer to the second exterior polygon. If there are interior polygons associated with it, Element 2, Ring 2 refers to the first interior polygon of the second exterior polygon.
- **Edges:** An edge refers to a line segment between two coordinates. Edge 1 refers to the segment between coordinate 1 and coordinate 2, Edge 2 refers to the line segment between coordinates 2 and 3, and so on. The most common place to see edge errors when validating geometries is with self-intersecting polygons. (The Open Geospatial Consortium simple features specification does not allow a polygon to self-intersect.) In such cases, Oracle reports error 13349 (polygon boundary crosses itself), including the Element, Ring, and Edge numbers where self-intersection occurs.

Examples

The following example validates a geometry (deliberately created as invalid) named `cola_invalid_geom`.

```
-- Validate; provide context if invalid
SELECT c.name, SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(c.shape, 0.005)
FROM cola_markets c WHERE c.name = 'cola_invalid_geom';
```

NAME

```
-----
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(C.SHAPE,0.005)
-----
```

```
cola_invalid_geom
13349 [Element <1>] [Ring <1>] [Edge <1>] [Edge <3>]
```

In the output for this example, 13349 indicates the error `ORA-13349: polygon boundary crosses itself`. The first ring of the first element has edges that intersect. The edges that intersect are edge 1 (the first and second vertices) and edge 3 (the third and fourth vertices).

Related Topics

- [SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT](#)

SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT

Format

```
SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT(
    geom_table      IN VARCHAR2,
    geom_column     IN VARCHAR2,
    result_table    IN VARCHAR2
    [, commit_interval IN NUMBER]);
```

Description

Examines a geometry column to determine if the stored geometries follow the defined rules for geometry objects, and returns context information about any invalid geometries.

Parameters

geom_table

Spatial geometry table.

geom_column

Geometry object column to be examined.

result_table

Result table to hold the validation results. A row is added to `result_table` for each invalid geometry. If there are no invalid geometries, one or more (depending on the `commit_interval` value) rows with a result of DONE are added.

commit_interval

Number of geometries to validate before Spatial performs an internal commit operation and writes a row with a result of DONE to `result_table` (if no rows for invalid geometries have been written since the last commit operation). If `commit_interval` is not specified, no internal commit operations are performed during the validation.

The `commit_interval` option is helpful if you want to look at the contents of `result_table` while the validation is in progress.

Usage Notes

This procedure loads the result table with validation results.

An empty result table (`result_table` parameter) must be created before calling this procedure. The format of the result table is: (sdo_rowid ROWID, result VARCHAR2(2000)). If `result_table` is not empty, you should truncate the table before calling the procedure; otherwise, the procedure appends rows to the existing data in the table.

The result table contains one row for each invalid geometry. A row is not written if a geometry is valid, except as follows:

- If `commit_interval` is not specified (or if the `commit_interval` value is greater than the number of geometries in the layer) and no invalid geometries are found, a single row with a RESULT value of DONE is written.

- If `commit_interval` is specified and if no invalid geometries are found between an internal commit and the previous internal commit (or start of validation for the first internal commit), a single row with the primary key of the last geometry validated and a `RESULT` value of `DONE` is written. (If there have been no invalid geometries since the last internal commit operation, this row replaces the previous row that had a result of `DONE`.)

In each row for an invalid geometry, the `SDO_ROWID` column contains the ROWID value of the row containing the invalid geometry, and the `RESULT` column contains an Oracle error message number and the context of the error (the coordinate, edge, or ring that causes the geometry to be invalid). You can then look up the error message for more information about the cause of the failure.

This procedure performs the following checks on each geometry in the layer (`geom_column`):

- All type consistency and geometry consistency checks that are performed by the [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#) function (see the Usage Notes for that function).
- If 4-digit `SDO_GTYPE` values are used, the geometry's `SDO_GTYPE` specifies the same dimensionality as specified in the applicable `DIMINFO` column value (from the `USER_SDO_GEOM_METADATA` view, which is described in [Section 2.6](#)).
- The geometry's `SRID` value (coordinate system) is the same as the one specified in the applicable `DIMINFO` column value (from the `USER_SDO_GEOM_METADATA` view, which is described in [Section 2.6](#)).

Examples

The following example validates the geometry objects stored in the `SHAPE` column of the `COLA_MARKETS` table. The example includes the creation of the result table. For this example, a deliberately invalid geometry was inserted into the table before the validation was performed.

```
-- Is a layer valid? (First, create the result table.)
CREATE TABLE val_results (sdo_rowid ROWID, result varchar2(1000));
-- (Next statement must be on one command line.)
CALL SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT('COLA_MARKETS', 'SHAPE', 'VAL_RESULTS');
```

Call completed.

```
SQL> SELECT * from val_results;
```

```
SDO_ROWID
```

```
-----
```

```
RESULT
```

```
-----
```

```
Rows Processed <12>
```

```
AAABXNAABAAK+YAAC
```

```
13349 [Element <1>] [Ring <1>] [Edge <1>] [Edge <3>]
```

Related Topics

- [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#)

SDO_GEOM.WITHIN_DISTANCE

Format

```
SDO_GEOM.WITHIN_DISTANCE(  
    geom1 IN SDO_GEOMETRY,  
    dim1  IN SDO_DIM_ARRAY,  
    dist  IN NUMBER,  
    geom2 IN SDO_GEOMETRY,  
    dim2  IN SDO_DIM_ARRAY  
    [, units IN VARCHAR2]  
    ) RETURN VARCHAR2;
```

or

```
SDO_GEOM.WITHIN_DISTANCE(  
    geom1 IN SDO_GEOMETRY,  
    dist  IN NUMBER,  
    geom2 IN SDO_GEOMETRY,  
    tol   IN NUMBER  
    [, units IN VARCHAR2]  
    ) RETURN VARCHAR2;
```

Description

Determines if two spatial objects are within some specified distance from each other.

Parameters

geom1

Geometry object.

dim1

Dimensional information array corresponding to `geom1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

dist

Distance value.

geom2

Geometry object.

dim2

Dimensional information array corresponding to `geom2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tol

Tolerance value (see [Section 1.5.5](#)).

units

Unit of measurement: a quoted string with `unit=` and an `SDO_UNIT` value from the `MDSYS.SDO_AREA_UNITS` table (for example, `'unit=KM'`). See [Section 2.8](#) for more information about unit of measurement specification.

If this parameter is not specified, the unit of measurement associated with the data is assumed. For geodetic data, the default unit of measurement is meters.

Usage Notes

For better performance, use the [SDO_WITHIN_DISTANCE](#) operator (described in [Chapter 11](#)) instead of the `SDO_GEOM.WITHIN_DISTANCE` function. For more information about performance considerations with operators and functions, see [Section 1.9](#).

This function returns `TRUE` for object pairs that are within the specified distance, and `FALSE` otherwise.

The distance between two extended objects (for example, nonpoint objects such as lines and polygons) is defined as the minimum distance between these two objects. Thus the distance between two adjacent polygons is zero.

If the function format with `tol` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example checks if `cola_b` and `cola_d` are within 1 unit apart at the shortest distance between them. (The example uses the definitions and data from [Section 2.1](#).)

```
-- Are two geometries within 1 unit of distance apart?
SELECT SDO_GEOM.WITHIN_DISTANCE(c_b.shape, m.diminfo, 1,
      c_d.shape, m.diminfo)
   FROM cola_markets c_b, cola_markets c_d, user_sdo_geom_metadata m
  WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
     AND c_b.name = 'cola_b' AND c_d.name = 'cola_d';
```

```
SDO_GEOM.WITHIN_DISTANCE(C_B.SHAPE,M.DIMINFO,1,C_D.SHAPE,M.DIMINFO)
```

```
-----
TRUE
```

Related Topics

- [SDO_GEOM.SDO_DISTANCE](#)

SDO_LRS Package (Linear Referencing System)

The MDSYS.SDO_LRS package contains subprograms that create, modify, query, and convert linear referencing elements. These subprograms do not change the state of the database. Most LRS subprograms are functions.

To use the subprograms in this chapter, you must understand the linear referencing system (LRS) concepts and techniques described in [Chapter 7](#).

[Table 16-1](#) lists subprograms related to creating and editing geometric segments.

Table 16-1 Subprograms for Creating and Editing Geometric Segments

Subprogram	Description
SDO_LRS.DEFINE_GEOM_SEGMENT	Defines a geometric segment.
SDO_LRS.REDEFINE_GEOM_SEGMENT	Populates the measures of all shape points of a geometric segment based on the start and end measures, overriding any previously assigned measures between the start point and end point.
SDO_LRS.CLIP_GEOM_SEGMENT	Clips a geometric segment (synonym of SDO_LRS.DYNAMIC_SEGMENT).
SDO_LRS.DYNAMIC_SEGMENT	Clips a geometric segment (synonym of SDO_LRS.CLIP_GEOM_SEGMENT).
SDO_LRS.CONCATENATE_GEOM_SEGMENTS	Concatenates two geometric segments into one segment.
SDO_LRS.LRS_INTERSECTION	Returns an LRS geometry object that is the topological intersection (AND operation) of two geometry objects where one or both are LRS geometries.
SDO_LRS.OFFSET_GEOM_SEGMENT	Returns the geometric segment at a specified offset from a geometric segment.
SDO_LRS.SPLIT_GEOM_SEGMENT	Splits a geometric segment into two segments.
SDO_LRS.RESET_MEASURE	Sets all measures of a geometric segment, including the start and end measures, to null values, overriding any previously assigned measures.
SDO_LRS.SET_PT_MEASURE	Sets the measure value of a specified point.
SDO_LRS.REVERSE_MEASURE	Returns a new geometric segment by reversing the measure values, but not the direction, of the original geometric segment.

Table 16–1 (Cont.) Subprograms for Creating and Editing Geometric Segments

Subprogram	Description
SDO_LRS.TRANSLATE_MEASURE	Returns a new geometric segment by translating the original geometric segment (that is, shifting the start and end measures by a specified value).
SDO_LRS.REVERSE_GEOMETRY	Returns a new geometric segment by reversing the measure values and the direction of the original geometric segment.

Table 16–2 lists subprograms related to querying geometric segments.

Table 16–2 Subprograms for Querying and Validating Geometric Segments

Subprogram	Description
SDO_LRS.VALID_GEOM_SEGMENT	Checks if a geometric segment is valid.
SDO_LRS.VALID_LRS_PT	Checks if an LRS point is valid.
SDO_LRS.VALID_MEASURE	Checks if a measure falls within the measure range of a geometric segment.
SDO_LRS.CONNECTED_GEOM_SEGMENTS	Checks if two geometric segments are spatially connected.
SDO_LRS.GEOM_SEGMENT_LENGTH	Returns the length of a geometric segment.
SDO_LRS.GEOM_SEGMENT_START_PT	Returns the start point of a geometric segment.
SDO_LRS.GEOM_SEGMENT_END_PT	Returns the end point of a geometric segment.
SDO_LRS.GEOM_SEGMENT_START_MEASURE	Returns the start measure of a geometric segment.
SDO_LRS.GEOM_SEGMENT_END_MEASURE	Returns the end measure of a geometric segment.
SDO_LRS.GET_MEASURE	Returns the measure of an LRS point.
SDO_LRS.GET_NEXT_SHAPE_PT	Returns the next shape point on a geometric segment after a specified measure value or LRS point.
SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE	Returns the measure value of the next shape point on a geometric segment after a specified measure value or LRS point.
SDO_LRS.GET_PREV_SHAPE_PT	Returns the previous shape point on a geometric segment before a specified measure value or LRS point.
SDO_LRS.GET_PREV_SHAPE_PT_MEASURE	Returns the measure value of the previous shape point on a geometric segment before a specified measure value or LRS point.
SDO_LRS.IS_GEOM_SEGMENT_DEFINED	Checks if an LRS segment is defined correctly.
SDO_LRS.IS_MEASURE_DECREASING	Checks if the measure values along an LRS segment are decreasing (that is, descending in numerical value).
SDO_LRS.IS_MEASURE_INCREASING	Checks if the measure values along an LRS segment are increasing (that is, ascending in numerical value).

Table 16–2 (Cont.) Subprograms for Querying and Validating Geometric Segments

Subprogram	Description
SDO_LRS.IS_SHAPE_PT_MEASURE	Checks if a specified measure value is associated with a shape point on a geometric segment.
SDO_LRS.MEASURE_RANGE	Returns the measure range of a geometric segment, that is, the difference between the start measure and end measure.
SDO_LRS.MEASURE_TO_PERCENTAGE	Returns the percentage (0 to 100) that a specified measure is of the measure range of a geometric segment.
SDO_LRS.PERCENTAGE_TO_MEASURE	Returns the measure value of a specified percentage (0 to 100) of the measure range of a geometric segment.
SDO_LRS.LOCATE_PT	Returns the point located at a specified distance from the start of a geometric segment.
SDO_LRS.PROJECT_PT	Returns the projection point of a specified point. The projection point is on the geometric segment.
SDO_LRS.FIND_LRS_DIM_POS	Returns the position of the measure dimension within the SDO_DIM_ARRAY structure for a specified SDO_GEOMETRY column.
SDO_LRS.FIND_MEASURE	Returns the measure of the closest point on a segment to a specified projection point.
SDO_LRS.FIND_OFFSET	Returns the signed offset (shortest distance) from a point to a geometric segment.
SDO_LRS.VALIDATE_LRS_GEOMETRY	Checks if an LRS geometry is valid.

Table 16–3 lists subprograms related to converting geometric segments.

Table 16–3 Subprograms for Converting Geometric Segments

Subprogram	Description
SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY	Converts a standard dimensional array to an LRS dimensional array by creating a measure dimension.
SDO_LRS.CONVERT_TO_LRS_GEOM	Converts a standard SDO_GEOMETRY line string to an LRS geometric segment by adding measure information.
SDO_LRS.CONVERT_TO_LRS_LAYER	Converts all geometry objects in a column of type SDO_GEOMETRY from standard line string geometries without measure information to LRS geometric segments with measure information, and updates the metadata.
SDO_LRS.CONVERT_TO_STD_DIM_ARRAY	Converts an LRS dimensional array to a standard dimensional array by removing the measure dimension.
SDO_LRS.CONVERT_TO_STD_GEOM	Converts an LRS geometric segment to a standard SDO_GEOMETRY line string by removing measure information.

Table 16–3 (Cont.) Subprograms for Converting Geometric Segments

Subprogram	Description
SDO_LRS.CONVERT_TO_STD_LAYER	Converts all geometry objects in a column of type SDO_GEOMETRY from LRS geometric segments with measure information to standard line string geometries without measure information, and updates the metadata.

For more information about conversion subprograms, see [Section 7.5.10](#).

The rest of this chapter provides reference information on the subprograms, listed in alphabetical order.

SDO_LRS.CLIP_GEOM_SEGMENT

Format

```
SDO_LRS.CLIP_GEOM_SEGMENT(
    geom_segment IN SDO_GEOMETRY,
    start_measure IN NUMBER,
    end_measure   IN NUMBER,
    tolerance     IN NUMBER DEFAULT 1.0e-8
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.CLIP_GEOM_SEGMENT(
    geom_segment IN SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    start_measure IN NUMBER,
    end_measure   IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns the geometry object resulting from a clip operation on a geometric segment.

Note: `SDO_LRS.CLIP_GEOM_SEGMENT` and [SDO_LRS.DYNAMIC_SEGMENT](#) are synonyms: both functions have the same parameters, behavior, and return value.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

start_measure

Start measure of the geometric segment.

end_measure

End measure of the geometric segment.

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

Usage Notes

An exception is raised if `geom_segment`, `start_measure`, or `end_measure` is invalid.

`start_measure` and `end_measure` can be any points on the geometric segment. They do not have to be in any specific order. For example, `start_measure` and `end_measure` can be 5 and 10, respectively, or 10 and 5, respectively.

The direction and measures of the resulting geometric segment are preserved (that is, they reflect the original segment).

The *_3D* format of this function (`SDO_LRS.CLIP_GEOM_SEGMENT_3D`) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

For more information about clipping geometric segments, see [Section 7.5.3](#).

Examples

The following example clips the geometric segment representing Route 1, returning the segment from measures 5 through 10. This segment might represent a construction zone. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.CLIP_GEOM_SEGMENT(route_geometry, 5, 10)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.CLIP_GEOM_SEGMENT(ROUTE_GEOMETRY,5,10) (SDO_GTYPE, SDO_SRID, SDO_POINT(X,
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
5, 4, 5, 8, 4, 8, 10, 4, 10))
```

SDO_LRS.CONCATENATE_GEOM_SEGMENTS

Format

```
SDO_LRS.CONCATENATE_GEOM_SEGMENTS(  
    geom_segment_1 IN SDO_GEOMETRY,  
    geom_segment_2 IN SDO_GEOMETRY,  
    tolerance      IN NUMBER DEFAULT 1.0e-8  
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.CONCATENATE_GEOM_SEGMENTS(  
    geom_segment_1 IN SDO_GEOMETRY,  
    dim_array_1    IN SDO_DIM_ARRAY,  
    geom_segment_2 IN SDO_GEOMETRY,  
    dim_array_2    IN SDO_DIM_ARRAY  
) RETURN SDO_GEOMETRY;
```

Description

Returns the geometry object resulting from the concatenation of two geometric segments.

Parameters

geom_segment_1

First geometric segment to be concatenated.

dim_array_1

Dimensional information array corresponding to `geom_segment_1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom_segment_2

Second geometric segment to be concatenated.

dim_array_2

Dimensional information array corresponding to `geom_segment_2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

Usage Notes

An exception is raised if `geom_segment_1` or `geom_segment_2` has an invalid geometry type or dimensionality, or if `geom_segment_1` and `geom_segment_2` are based on different coordinate systems.

The direction of the first geometric segment is preserved, and all measures of the second segment are shifted so that its start measure is the same as the end measure of the first segment.

The geometry type of `geom_segment_1` and `geom_segment_2` must be line or multiline. Neither can be a polygon.

The `_3D` format of this function (`SDO_LRS.CONCATENATE_GEOM_SEGMENTS_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

For more information about concatenating geometric segments, see [Section 7.5.5](#).

Examples

The following example defines the geometric segment, splits it into two segments, then concatenates those segments. (This example uses the definitions from the example in [Section 7.7](#). The definitions of `result_geom_1`, `result_geom_2`, and `result_geom_3` are displayed in [Example 7-3](#).)

```
DECLARE
geom_segment SDO_GEOMETRY;
line_string SDO_GEOMETRY;
dim_array SDO_DIM_ARRAY;
result_geom_1 SDO_GEOMETRY;
result_geom_2 SDO_GEOMETRY;
result_geom_3 SDO_GEOMETRY;

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
  WHERE a.route_name = 'Route1';
SELECT m.diminfo into dim_array from
  user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

-- Define the LRS segment for Route1.
SDO_LRS.DEFINE_GEOM_SEGMENT (geom_segment,
  dim_array,
  0, -- Zero starting measure: LRS segment starts at start of route.
  27); -- End of LRS segment is at measure 27.

SELECT a.route_geometry INTO line_string FROM lrs_routes a
  WHERE a.route_name = 'Route1';

-- Split Route1 into two segments.
SDO_LRS.SPLIT_GEOM_SEGMENT(line_string,dim_array,5,result_geom_1,result_geom_2);

-- Concatenate the segments that were just split.
result_geom_3 := SDO_LRS.CONCATENATE_GEOM_SEGMENTS(result_geom_1, dim_array,
result_geom_2, dim_array);

-- Insert geometries into table, to display later.
INSERT INTO lrs_routes VALUES(
  11,
  'result_geom_1',
  result_geom_1
);
INSERT INTO lrs_routes VALUES(
  12,
  'result_geom_2',
  result_geom_2
);
INSERT INTO lrs_routes VALUES(
  13,
  'result_geom_3',
```



```
    result_geom_3  
);  
  
END;  
/
```

SDO_LRS.CONNECTED_GEOM_SEGMENTS

Format

```
SDO_LRS.CONNECTED_GEOM_SEGMENTS(  
  geom_segment_1 IN SDO_GEOMETRY,  
  geom_segment_2 IN SDO_GEOMETRY,  
  tolerance      IN NUMBER DEFAULT 1.0e-8  
) RETURN VARCHAR2;
```

or

```
SDO_LRS.CONNECTED_GEOM_SEGMENTS(  
  geom_segment_1 IN SDO_GEOMETRY,  
  dim_array_1    IN SDO_DIM_ARRAY,  
  geom_segment_2 IN SDO_GEOMETRY,  
  dim_array_2    IN SDO_DIM_ARRAY  
) RETURN VARCHAR2;
```

Description

Checks if two geometric segments are spatially connected.

Parameters

geom_segment_1

First of two geometric segments to be checked.

dim_array_1

Dimensional information array corresponding to `geom_segment_1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom_segment_2

Second of two geometric segments to be checked.

dim_array_2

Dimensional information array corresponding to `geom_segment_2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

Usage Notes

This function returns TRUE if the geometric segments are spatially connected and FALSE if the geometric segments are not spatially connected.

An exception is raised if `geom_segment_1` or `geom_segment_2` has an invalid geometry type or dimensionality, or if `geom_segment_1` and `geom_segment_2` are based on different coordinate systems.

The `_3D` format of this function (`SDO_LRS.CONNECTED_GEOM_SEGMENTS_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example checks if two geometric segments (results of a previous split operation) are spatially connected.

```
-- Are result_geom_1 and result_geom2 connected?
SELECT  SDO_LRS.CONNECTED_GEOM_SEGMENTS(a.route_geometry,
      b.route_geometry, 0.005)
  FROM lrs_routes a, lrs_routes b
  WHERE a.route_id = 11 AND b.route_id = 12;

SDO_LRS.CONNECTED_GEOM_SEGMENTS(A.ROUTE_GEOMETRY,B.ROUTE_GEOMETRY,0.005)
-----
TRUE
```

SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY

Format

```
SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY(  
    dim_array      IN SDO_DIM_ARRAY  
    [, lower_bound IN NUMBER,  
    upper_bound   IN NUMBER,  
    tolerance     IN NUMBER]  
    ) RETURN SDO_DIM_ARRAY;
```

or

```
SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY(  
    dim_array      IN SDO_DIM_ARRAY,  
    dim_name       IN VARCHAR2  
    [, lower_bound IN NUMBER,  
    upper_bound   IN NUMBER,  
    tolerance     IN NUMBER]  
    ) RETURN SDO_DIM_ARRAY;
```

or

```
SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY(  
    dim_array      IN SDO_DIM_ARRAY,  
    dim_name       IN VARCHAR2,  
    dim_pos        IN INTEGER  
    [, lower_bound IN NUMBER,  
    upper_bound   IN NUMBER,  
    tolerance     IN NUMBER]  
    ) RETURN SDO_DIM_ARRAY;
```

Description

Converts a standard dimensional array to an LRS dimensional array by creating a measure dimension.

Parameters

dim_array

Dimensional information array corresponding to the layer (column of geometries) to be converted, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in [Section 2.6](#)).

dim_name

Name of the measure dimension (M, if not otherwise specified).

dim_pos

Position of the measure dimension (the last SDO_DIM_ELEMENT object position in the SDO_DIM_ARRAY, if not otherwise specified).

lower_bound

Lower bound (SDO_LB value in the SDO_DIM_ELEMENT definition) of the ordinate in the measure dimension.

upper_bound

Upper bound (SDO_UB value in the SDO_DIM_ELEMENT definition) of the ordinate in the measure dimension.

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

Usage Notes

This function converts a standard dimensional array to an LRS dimensional array by creating a measure dimension. Specifically, it adds an SDO_DIM_ELEMENT object at the end of the current SDO_DIM_ELEMENT objects in the SDO_DIM_ARRAY for the dimensional array (unless another dim_pos is specified), and sets the SDO_DIMNAME value in this added SDO_DIM_ELEMENT to M (unless another dim_name is specified). It sets the other values in the added SDO_DIM_ELEMENT according to the values of the upper_bound, lower_bound, and tolerance parameter values.

If dim_array already contains dimensional information, the dim_array is returned.

The *_3D* format of this function (SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY_3D) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

For more information about conversion functions, see [Section 7.5.10](#).

Examples

The following example converts the dimensional array for the LRS_ROUTES table to LRS format. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY(m.diminfo)
   FROM user_sdo_geom_metadata m
   WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

SDO_LRS.CONVERT_TO_LRS_DIM_ARRAY(M.DIMINFO) (SDO_DIMNAME, SDO_LB, SDO_UB, SDO_TOL
-----
SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X', 0, 20, .005), SDO_DIM_ELEMENT('Y', 0, 20, .00
5), SDO_DIM_ELEMENT('M', 0, 20, .005))
```

SDO_LRS.CONVERT_TO_LRS_GEOM

Format

```
SDO_LRS.CONVERT_TO_LRS_GEOM(  
    standard_geom IN SDO_GEOMETRY  
    [, start_measure IN NUMBER,  
    end_measure IN NUMBER]  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.CONVERT_TO_LRS_GEOM(  
    standard_geom IN SDO_GEOMETRY,  
    dim_array IN SDO_DIM_ARRAY  
    [, start_measure IN NUMBER,  
    end_measure IN NUMBER]  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.CONVERT_TO_LRS_GEOM(  
    standard_geom IN SDO_GEOMETRY,  
    m_pos IN INTEGER  
    [, start_measure IN NUMBER,  
    end_measure IN NUMBER]  
    ) RETURN SDO_GEOMETRY;
```

Description

Converts a standard SDO_GEOMETRY line string to an LRS geometric segment by adding measure information.

Parameters

standard_geom

Line string geometry that does not contain measure information.

dim_array

Dimensional information array corresponding to `standard_geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

m_pos

Position of the measure dimension. If specified, must be 3 or 4. By default, the measure dimension is the last dimension in the SDO_DIM_ARRAY.

start_measure

Distance measured from the start point of a geometric segment to the start point of the linear feature. The default is 0.

end_measure

Distance measured from the end point of a geometric segment to the start point of the linear feature. The default is the cartographic length (for example, 75 if the cartographic length is 75 and the unit of measure is miles).

Usage Notes

This function returns an LRS geometric segment with measure information, with measure information provided for all shape points.

An exception is raised if `standard_geom` has an invalid geometry type or dimensionality, if `m_pos` is less than 3 or greater than 4, or if `start_measure` or `end_measure` is out of range.

The *_3D* format of this function (`SDO_LRS.CONVERT_TO_LRS_GEOM_3D`) is available; however, the `m_pos` parameter is not available for `SDO_LRS.CONVERT_TO_LRS_GEOM_3D`. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

For more information about conversion functions, see [Section 7.5.10](#).

Examples

The following example converts the geometric segment representing Route 1 to LRS format. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.CONVERT_TO_LRS_GEOM(a.route_geometry, m.diminfo)
   FROM lrs_routes a, user_sdo_geom_metadata m
   WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
        AND a.route_id = 1;

SDO_LRS.CONVERT_TO_LRS_GEOM(A.ROUTE_GEOMETRY,M.DIMINFO) (SDO_GTYPE, SDO_SRID, SDO
-----
SDO_GEOMETRY(3002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, NULL, 8, 10, 22, 5, 14, 27))
```

SDO_LRS.CONVERT_TO_LRS_LAYER

Format

```
SDO_LRS.CONVERT_TO_LRS_LAYER(  
    table_name    IN VARCHAR2,  
    column_name   IN VARCHAR2  
    [, lower_bound IN NUMBER,  
    upper_bound   IN NUMBER,  
    tolerance     IN NUMBER]  
    ) RETURN VARCHAR2;
```

or

```
SDO_LRS.CONVERT_TO_LRS_LAYER(  
    table_name    IN VARCHAR2,  
    column_name   IN VARCHAR2,  
    dim_name      IN VARCHAR2,  
    dim_pos       IN INTEGER  
    [, lower_bound IN NUMBER,  
    upper_bound   IN NUMBER,  
    tolerance     IN NUMBER]  
    ) RETURN VARCHAR2;
```

Description

Converts all geometry objects in a column of type SDO_GEOMETRY (that is, converts a layer) from standard line string geometries without measure information to LRS geometric segments with measure information, and updates the metadata in the USER_SDO_GEOM_METADATA view.

Parameters

table_name

Table containing the column with the SDO_GEOMETRY objects.

column_name

Column in `table_name` containing the SDO_GEOMETRY objects.

dim_name

Name of the measure dimension. If this parameter is null, M is assumed.

dim_pos

Position of the measure dimension within the SDO_DIM_ARRAY structure for the specified SDO_GEOMETRY column. If this parameter is null, the number corresponding to the last position is assumed.

lower_bound

Lower bound (SDO_LB value in the SDO_DIM_ELEMENT definition) of the ordinate in the measure dimension.

upper_bound

Upper bound (SDO_UB value in the SDO_DIM_ELEMENT definition) of the ordinate in the measure dimension.

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

Usage Notes

This function returns TRUE if the conversion was successful or if the layer already contains measure information, and the function returns an exception if the conversion was not successful.

An exception is raised if the existing dimensional information for the table is invalid.

The measure values are assigned based on a start measure of zero and an end measure of the cartographic length.

If a spatial index already exists on `column_name`, you must delete (drop) the index before converting the layer and create a new index after converting the layer. For information about deleting and creating indexes, see the [DROP INDEX](#) and [CREATE INDEX](#) statements in [Chapter 10](#).

The `_3D` format of this function (SDO_LRS.CONVERT_TO_LRS_LAYER_3D) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

For more information about conversion functions, see [Section 7.5.10](#).

Examples

The following example converts the geometric segments in the ROUTE_GEOMETRY column of the LRS_ROUTES table to LRS format. (This example uses the definitions from the example in [Section 7.7](#).) The SELECT statement shows that dimensional information has been added (that is, SDO_DIM_ELEMENT('M', NULL, NULL, NULL) is included in the definition).

```
BEGIN
  IF (SDO_LRS.CONVERT_TO_LRS_LAYER('LRS_ROUTES', 'ROUTE_GEOMETRY') = 'TRUE')
    THEN
      DBMS_OUTPUT.PUT_LINE('Conversion from STD_LAYER to LRS_LAYER succeeded.');
```

```
    ELSE
      DBMS_OUTPUT.PUT_LINE('Conversion from STD_LAYER to LRS_LAYER failed.');
```

```
  END IF;
END;
```

```
.
```

```
/
```

```
Conversion from STD_LAYER to LRS_LAYER succeeded.
```

PL/SQL procedure successfully completed.

```
SQL> SELECT diminfo FROM user_sdo_geom_metadata WHERE table_name = 'LRS_ROUTES'
AND column_name = 'ROUTE_GEOMETRY';
```

```
DIMINFO(SDO_DIMNAME, SDO_LB, SDO_UB, SDO_TOLERANCE)
```

```
-----
SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X', 0, 20, .005), SDO_DIM_ELEMENT('Y', 0, 20, .005), SDO_DIM_ELEMENT('M', NULL, NULL, NULL))
```

SDO_LRS.CONVERT_TO_STD_DIM_ARRAY

Format

```
SDO_LRS.CONVERT_TO_STD_DIM_ARRAY(
    dim_array IN SDO_DIM_ARRAY
    [, m_pos   IN INTEGER]
    ) RETURN SDO_DIM_ARRAY;
```

Description

Converts an LRS dimensional array to a standard dimensional array by removing the measure dimension.

Parameters

dim_array

Dimensional information array corresponding to the layer (column of geometries) to be converted, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in [Section 2.6](#)).

m_pos

Position of the measure dimension. If specified, must be 3 or 4. By default, the measure dimension is the last dimension in the SDO_DIM_ARRAY.

Usage Notes

This function converts an LRS dimensional array to a standard dimensional array by removing the measure dimension. Specifically, it removes the SDO_DIM_ELEMENT object at the end of the current SDO_DIM_ELEMENT objects in the SDO_DIM_ARRAY for the dim_array.

An exception is raised if m_pos is invalid (less than 3 or greater than 4).

If dim_array is already a standard dimensional array (that is, does not contain dimensional information), the dim_array is returned.

The *_3D* format of this function (SDO_LRS.CONVERT_TO_STD_DIM_ARRAY_3D) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

For more information about conversion functions, see [Section 7.5.10](#).

Examples

The following example converts the dimensional array for the LRS_ROUTES table to standard format. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.CONVERT_TO_STD_DIM_ARRAY(m.diminfo)
       FROM user_sdo_geom_metadata m
       WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

SDO_LRS.CONVERT_TO_STD_DIM_ARRAY(M.DIMINFO) (SDO_DIMNAME, SDO_LB, SDO_UB, SDO_TOL
-----
SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X', 0, 20, .005), SDO_DIM_ELEMENT('Y', 0, 20, .00
5))
```

SDO_LRS.CONVERT_TO_STD_GEOM

Format

```
SDO_LRS.CONVERT_TO_STD_GEOM(
  lrs_geom IN SDO_GEOMETRY
  [, dim_array IN SDO_DIM_ARRAY]
) RETURN SDO_GEOMETRY;
```

Description

Converts an LRS geometric segment to a standard SDO_GEOMETRY line string by removing measure information.

Parameters

lrs_geom

LRS geometry that contains measure information.

dim_array

Dimensional information array corresponding to `lrs_geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns an SDO_GEOMETRY object in which all measure information is removed.

The `_3D` format of this function (`SDO_LRS.CONVERT_TO_STD_GEOM_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

For more information about conversion functions, see [Section 7.5.10](#).

Examples

The following example converts the geometric segment representing Route 1 to standard format. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.CONVERT_TO_STD_GEOM(a.route_geometry, m.diminfo)
  FROM lrs_routes a, user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
  AND a.route_id = 1;

SDO_LRS.CONVERT_TO_STD_GEOM(A.ROUTE_GEOMETRY,M.DIMINFO) (SDO_GTYPE, SDO_SRID, SDO
-----
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 2, 4, 8, 4, 12, 4, 12, 10, 8, 10, 5, 14))
```

SDO_LRS.CONVERT_TO_STD_LAYER

Format

```
SDO_LRS.CONVERT_TO_STD_LAYER(  
    table_name    IN VARCHAR2,  
    column_name  IN VARCHAR2  
    ) RETURN VARCHAR2;
```

Description

Converts all geometry objects in a column of type SDO_GEOMETRY (that is, converts a layer) from LRS geometric segments with measure information to standard line string geometries without measure information, and updates the metadata in the USER_SDO_GEOM_METADATA view.

Parameters

table_name

Table containing the column with the SDO_GEOMETRY objects.

column_name

Column in `table_name` containing the SDO_GEOMETRY objects.

Usage Notes

This function returns TRUE if the conversion was successful or if the layer already is a standard layer (that is, contains geometries without measure information), and the function returns an exception if the conversion was not successful.

If a spatial index already exists on `column_name`, you must delete (drop) the index before converting the layer and create a new index after converting the layer. For information about deleting and creating indexes, see the [DROP INDEX](#) and [CREATE INDEX](#) statements in [Chapter 10](#).

The `_3D` format of this function (`SDO_LRS.CONVERT_TO_STD_LAYER_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

For more information about conversion functions, see [Section 7.5.10](#).

Examples

The following example converts the geometric segments in the ROUTE_GEOMETRY column of the LRS_ROUTES table to standard format. (This example uses the definitions from the example in [Section 7.7](#).) The SELECT statement shows that dimensional information has been removed (that is, no SDO_DIM_ELEMENT('M', NULL, NULL, NULL) is included in the definition).

```
BEGIN  
    IF (SDO_LRS.CONVERT_TO_STD_LAYER('LRS_ROUTES', 'ROUTE_GEOMETRY') = 'TRUE')  
        THEN  
            DBMS_OUTPUT.PUT_LINE('Conversion from LRS_LAYER to STD_LAYER succeeded.');
```

```
        ELSE  
            DBMS_OUTPUT.PUT_LINE('Conversion from LRS_LAYER to STD_LAYER failed.');
```

```
        END IF;
```

```
END;
```

```
.  
/  
Conversion from LRS_LAYER to STD_LAYER succeeded.  
  
PL/SQL procedure successfully completed.  
  
SELECT diminfo FROM user_sdo_geom_metadata  
  WHERE table_name = 'LRS_ROUTES' AND column_name = 'ROUTE_GEOMETRY';  
  
DIMINFO(SDO_DIMNAME, SDO_LB, SDO_UB, SDO_TOLERANCE)  
-----  
SDO_DIM_ARRAY(SDO_DIM_ELEMENT('X', 0, 20, .005), SDO_DIM_ELEMENT('Y', 0, 20, .00  
5))
```

SDO_LRS.DEFINE_GEOM_SEGMENT

Format

```
SDO_LRS.DEFINE_GEOM_SEGMENT(  
    geom_segment IN OUT SDO_GEOMETRY  
    [, start_measure IN NUMBER,  
    end_measure IN NUMBER]);
```

or

```
SDO_LRS.DEFINE_GEOM_SEGMENT(  
    geom_segment IN OUT SDO_GEOMETRY,  
    dim_array IN SDO_DIM_ARRAY  
    [, start_measure IN NUMBER,  
    end_measure IN NUMBER]);
```

Description

Defines a geometric segment by assigning start and end measures to a geometric segment, and assigns values to any null measures.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

start_measure

Distance measured from the start point of a geometric segment to the start point of the linear feature. The default is the existing value (if any) in the measure dimension; otherwise, the default is 0.

end_measure

Distance measured from the end point of a geometric segment to the start point of the linear feature. The default is the existing value (if any) in the measure dimension; otherwise, the default is the cartographic length of the segment.

Usage Notes

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality, or if `start_measure` or `end_measure` is out of range.

All unassigned measures of the geometric segment will be populated automatically.

To store the resulting geometric segment (`geom_segment`) in the database, you must execute an UPDATE or INSERT statement, as appropriate.

The `_3D` format of this procedure (`SDO_LRS.DEFINE_GEOM_SEGMENT_3D`) is available. For information about `_3D` formats of LRS functions and procedures, see [Section 7.4](#).

For more information about defining a geometric segment, see [Section 7.5.1](#).

Examples

The following example defines the geometric segment, splits it into two segments, then concatenates those segments. (This example uses the definitions from the example in [Section 7.7](#). The definitions of `result_geom_1`, `result_geom_2`, and `result_geom_3` are displayed in [Example 7-3](#).)

```

DECLARE
geom_segment SDO_GEOMETRY;
line_string SDO_GEOMETRY;
dim_array SDO_DIM_ARRAY;
result_geom_1 SDO_GEOMETRY;
result_geom_2 SDO_GEOMETRY;
result_geom_3 SDO_GEOMETRY;

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
  WHERE a.route_name = 'Route1';
SELECT m.diminfo into dim_array from
  user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

-- Define the LRS segment for Route1. This will populate any null measures.
SDO_LRS.DEFINE_GEOM_SEGMENT (geom_segment,
  dim_array,
  0,    -- Zero starting measure: LRS segment starts at start of route.
  27); -- End of LRS segment is at measure 27.

SELECT a.route_geometry INTO line_string FROM lrs_routes a
  WHERE a.route_name = 'Route1';

-- Split Route1 into two segments.
SDO_LRS.SPLIT_GEOM_SEGMENT(line_string,dim_array,5,result_geom_1,result_geom_2);

-- Concatenate the segments that were just split.
result_geom_3 := SDO_LRS.CONCATENATE_GEOM_SEGMENTS(result_geom_1, dim_array,
result_geom_2, dim_array);

-- Update and insert geometries into table, to display later.
UPDATE lrs_routes a SET a.route_geometry = geom_segment
  WHERE a.route_id = 1;

INSERT INTO lrs_routes VALUES(
  11,
  'result_geom_1',
  result_geom_1
);
INSERT INTO lrs_routes VALUES(
  12,
  'result_geom_2',
  result_geom_2
);
INSERT INTO lrs_routes VALUES(
  13,
  'result_geom_3',
  result_geom_3
);

```

```
END;  
/
```

SDO_LRS.DYNAMIC_SEGMENT

Format

```
SDO_LRS.DYNAMIC_SEGMENT(
    geom_segment IN SDO_GEOMETRY,
    start_measure IN NUMBER,
    end_measure   IN NUMBER,
    tolerance     IN NUMBER DEFAULT 1.0e-8
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.DYNAMIC_SEGMENT(
    geom_segment IN SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    start_measure IN NUMBER,
    end_measure   IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns the geometry object resulting from a clip operation on a geometric segment.

Note: [SDO_LRS.CLIP_GEOM_SEGMENT](#) and [SDO_LRS.DYNAMIC_SEGMENT](#) are synonyms: both functions have the same parameters, behavior, and return value.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

start_measure

Start measure of the geometric segment.

end_measure

End measure of the geometric segment.

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

Usage Notes

An exception is raised if `geom_segment`, `start_measure`, or `end_measure` is invalid.

The direction and measures of the resulting geometric segment are preserved.

For more information about clipping a geometric segment, see [Section 7.5.3](#).

Examples

The following example clips the geometric segment representing Route 1, returning the segment from measures 5 through 10. This segment might represent a construction zone. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.DYNAMIC_SEGMENT(route_geometry, 5, 10)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.DYNAMIC_SEGMENT(ROUTE_GEOMETRY,5,10) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
5, 4, 5, 8, 4, 8, 10, 4, 10))
```

SDO_LRS.FIND_LRS_DIM_POS

Format

```
SDO_LRS.FIND_LRS_DIM_POS(  
    table_name IN VARCHAR2,  
    column_name IN VARCHAR2  
) RETURN INTEGER;
```

Description

Returns the position of the measure dimension within the SDO_DIM_ARRAY structure for a specified SDO_GEOMETRY column.

Parameters

table_name

Table containing the column with the SDO_GEOMETRY objects.

column_name

Column in table_name containing the SDO_GEOMETRY objects.

Usage Notes

None.

Examples

The following example returns the position of the measure dimension within the SDO_DIM_ARRAY structure for geometries in the ROUTE_GEOMETRY column of the LRS_ROUTES table. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.FIND_LRS_DIM_POS('LRS_ROUTES', 'ROUTE_GEOMETRY') FROM DUAL;
```

```
SDO_LRS.FIND_LRS_DIM_POS('LRS_ROUTES', 'ROUTE_GEOMETRY')
```

```
-----
```

3

SDO_LRS.FIND_MEASURE

Format

```
SDO_LRS.FIND_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    point        IN SDO_GEOMETRY  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.FIND_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    point        IN SDO_GEOMETRY  
    ) RETURN NUMBER;
```

Description

Returns the measure of the closest point on a segment to a specified projection point.

Parameters

geom_segment

Cartographic representation of a linear feature. This function returns the measure of the point on this segment that is closest to the projection point.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

point

Projection point. This function returns the measure of the point on `geom_segment` that is closest to the projection point.

Usage Notes

This function returns the measure of the point on `geom_segment` that is closest to the projection point. For example, if the projection point represents a shopping mall, the function could be used to find how far from the start of the highway is the point on the highway that is closest to the shopping mall.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality, or if `geom_segment` and `point` are based on different coordinate systems.

The `_3D` format of this function (`SDO_LRS.FIND_MEASURE_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example finds the measure for the point on the geometric segment representing Route 1 that is closest to the point (10, 7). (This example uses the definitions from the example in [Section 7.7](#).)

```
-- Find measure for point on segment closest to 10,7.
-- Should return 15 (for point 12,7).
SELECT SDO_LRS.FIND_MEASURE(a.route_geometry, m.diminfo,
    SDO_GEOMETRY(3001, NULL, NULL,
        SDO_ELEM_INFO_ARRAY(1, 1, 1),
        SDO_ORDINATE_ARRAY(10, 7, NULL)) )
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
    AND a.route_id = 1;

SDO_LRS.FIND_MEASURE(A.ROUTE_GEOMETRY,M.DIMINFO,SDO_GEOMETRY(3001,NULL,NUL
```

15

SDO_LRS.FIND_OFFSET

Format

```
SDO_LRS.FIND_OFFSET(  
    geom_segment IN SDO_GEOMETRY,  
    point        IN SDO_GEOMETRY,  
    tolerance    IN NUMBER DEFAULT 1.0e-8  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.FIND_OFFSET(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    point        IN SDO_GEOMETRY  
    [, point_dim_array IN SDO_GEOMETRY]  
    ) RETURN NUMBER;
```

Description

Returns the signed offset (shortest distance) from a point to a geometric segment.

Parameters

geom_segment

Geometric segment to be checked for distance from `point`.

point

Point whose shortest distance from `geom_segment` is to be returned.

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

point_dim_array

Dimensional information array corresponding to `point`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function calls the [SDO_LRS.PROJECT_PT](#) function format that includes the `offset` output parameter: it passes in the geometric segment and point information, and it returns the [SDO_LRS.PROJECT_PT](#) `offset` parameter value. Thus, to find the offset of a point from a geometric segment, you can use either this function or the [SDO_LRS.PROJECT_PT](#) function with the `offset` parameter.

An exception is raised if `geom_segment` or `point` has an invalid geometry type or dimensionality, or if `geom_segment` and `point` are based on different coordinate systems.

For more information about offsets to a geometric segment, see [Section 7.1.5](#).

Examples

The following example returns the offset of point (9,3,NULL) from the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).) As you can see from [Figure 7-20](#) in [Section 7.7](#), the point at (9,3,NULL) is on the right side along the segment, and therefore the offset has a negative value, as explained in [Section 7.1.5](#). The point at (9,3,NULL) is one distance unit away from the point at (9,4,NULL), which is on the segment.

```
-- Find the offset of point (9,3,NULL) from the road; should return -1.
SELECT SDO_LRS.FIND_OFFSET(route_geometry,
    SDO_GEOMETRY(3301, NULL, NULL,
        SDO_ELEM_INFO_ARRAY(1, 1, 1),
        SDO_ORDINATE_ARRAY(9, 3, NULL)) )
FROM lrs_routes WHERE route_id = 1;

SDO_LRS.FIND_OFFSET(ROUTE_GEOMETRY,SDO_GEOMETRY(3301,NULL,NULL,SDO_ELEM_INFO_ARR
-----
-1
```

SDO_LRS.GEOM_SEGMENT_END_MEASURE

Format

```
SDO_LRS.GEOM_SEGMENT_END_MEASURE(
  geom_segment IN SDO_GEOMETRY
  [, dim_array  IN SDO_DIM_ARRAY]
) RETURN NUMBER;
```

Description

Returns the end measure of a geometric segment.

Parameters

geom_segment

Geometric segment whose end measure is to be returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns the end measure of `geom_segment`.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

The *_3D* format of this function (`SDO_LRS.GEOM_SEGMENT_END_MEASURE_3D`) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the end measure of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.GEOM_SEGMENT_END_MEASURE(route_geometry)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.GEOM_SEGMENT_END_MEASURE (ROUTE_GEOMETRY)
-----
```

27

SDO_LRS.GEOM_SEGMENT_END_PT

Format

```
SDO_LRS.GEOM_SEGMENT_END_PT(
    geom_segment IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
) RETURN SDO_GEOMETRY;
```

Description

Returns the end point of a geometric segment.

Parameters

geom_segment

Geometric segment whose end point is to be returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns the end point of `geom_segment`.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

The *_3D* format of this function (`SDO_LRS.GEOM_SEGMENT_END_PT_3D`) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the end point of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.GEOM_SEGMENT_END_PT(route_geometry)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.GEOM_SEGMENT_END_PT(ROUTE_GEOMETRY) (SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
5, 14, 27))
```

SDO_LRS.GEOM_SEGMENT_LENGTH

Format

```
SDO_LRS.GEOM_SEGMENT_LENGTH(
    geom_segment IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
) RETURN NUMBER;
```

Description

Returns the length of a geometric segment.

Parameters

geom_segment

Geometric segment whose length is to be calculated.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns the length of `geom_segment`. The length is the geometric length, which is not the same as the total of the measure unit values. To determine how long a segment is in terms of measure units, subtract the result of an [SDO_LRS.GEOM_SEGMENT_START_MEASURE](#) operation from the result of an [SDO_LRS.GEOM_SEGMENT_END_MEASURE](#) operation.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

The `_3D` format of this function (`SDO_LRS.GEOM_SEGMENT_LENGTH_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the length of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.GEOM_SEGMENT_LENGTH(route_geometry)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.GEOM_SEGMENT_LENGTH(ROUTE_GEOMETRY)
-----
```

27

SDO_LRS.GEOM_SEGMENT_START_MEASURE

Format

```
SDO_LRS.GEOM_SEGMENT_START_MEASURE(
    geom_segment IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
) RETURN NUMBER;
```

Description

Returns the start measure of a geometric segment.

Parameters

geom_segment

Geometric segment whose start measure is to be returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns the start measure of `geom_segment`.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

The *_3D* format of this function (`SDO_LRS.GEOM_SEGMENT_START_MEASURE_3D`) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the start measure of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT  SDO_LRS.GEOM_SEGMENT_START_MEASURE(route_geometry)
        FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.GEOM_SEGMENT_START_MEASURE(ROUTE_GEOMETRY)
```

```
-----
0
```

SDO_LRS.GEOM_SEGMENT_START_PT

Format

```
SDO_LRS.GEOM_SEGMENT_START_PT(
    geom_segment IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
    ) RETURN SDO_GEOMETRY;
```

Description

Returns the start point of a geometric segment.

Parameters

geom_segment

Geometric segment whose start point is to be returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns the start point of `geom_segment`.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

The *_3D* format of this function (`SDO_LRS.GEOM_SEGMENT_START_PT_3D`) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the start point of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT  SDO_LRS.GEOM_SEGMENT_START_PT(route_geometry)
        FROM lrs_routes WHERE route_id = 1;

SDO_LRS.GEOM_SEGMENT_START_PT(ROUTE_GEOMETRY)(SDO_GTYPE, SDO_SRID, SDO_POINT(X,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
2, 2, 0))
```

SDO_LRS.GET_MEASURE

Format

```
SDO_LRS.GET_MEASURE(
    point      IN SDO_GEOMETRY
    [, dim_array IN SDO_DIM_ARRAY]
) RETURN NUMBER;
```

Description

Returns the measure of an LRS point.

Parameters

point

Point whose measure is to be returned.

dim_array

Dimensional information array corresponding to `point`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns the measure of an LRS point.

If `point` is not valid, an "invalid LRS point" exception is raised.

Contrast this function with [SDO_LRS.PROJECT_PT](#), which accepts as input a point that is not necessarily on the geometric segment, but which returns a point that is on the geometric segment, as opposed to a measure value. As the following example shows, the `SDO_LRS.GET_MEASURE` function can be used to return the measure of the projected point returned by [SDO_LRS.PROJECT_PT](#).

The `_3D` format of this function (`SDO_LRS.GET_MEASURE_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the measure of a projected point. In this case, the point resulting from the projection is 9 units from the start of the segment.

```
SELECT SDO_LRS.GET_MEASURE (
    SDO_LRS.PROJECT_PT(a.route_geometry, m.diminfo,
        SDO_GEOMETRY(3001, NULL, NULL,
            SDO_ELEM_INFO_ARRAY(1, 1, 1),
            SDO_ORDINATE_ARRAY(9, 3, NULL)) ),
    m.diminfo )
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;
```

```
SDO_LRS.GET_MEASURE(SDO_LRS.PROJECT_PT(A.ROUTE_GEOMETRY,M.DIMINFO,SDO_GEOM
```

SDO_LRS.GET_NEXT_SHAPE_PT

Format

```
SDO_LRS.GET_NEXT_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    measure      IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.GET_NEXT_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    measure      IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.GET_NEXT_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    point        IN SDO_GEOMETRY  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.GET_NEXT_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    point        IN SDO_GEOMETRY  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns the next shape point on a geometric segment after a specified measure value or LRS point.

Parameters

geom_segment

Geometric segment.

measure

Measure value on the geometric segment for which to return the next shape point.

point

Point for which to return the next shape point. If `point` is not on `geom_segment`, the point on the geometric segment closest to the specified point is computed, and the next shape point after that point is returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

If `measure` or `point` identifies the end point of the geometric segment, a null value is returned.

An exception is raised if `measure` is not a valid value for `geom_segment` or if `point` is not a valid LRS point.

Contrast this function with [SDO_LRS.GET_PREV_SHAPE_PT](#), which returns the previous shape point on a geometric segment before a specified measure value or LRS point.

The `_3D` format of this function ([SDO_LRS.GET_NEXT_SHAPE_PT_3D](#)) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the next shape point after measure 14 on the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.GET_NEXT_SHAPE_PT(a.route_geometry, 14)
       FROM lrs_routes a WHERE a.route_id = 1;
```

```
SDO_LRS.GET_NEXT_SHAPE_PT(A.ROUTE_GEOMETRY,14) (SDO_GTYPE, SDO_SRID, SDO_POINT(X,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
12, 10, 18))
```

SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE

Format

```
SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    measure      IN NUMBER  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    measure      IN NUMBER  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    point        IN SDO_GEOMETRY  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    point        IN SDO_GEOMETRY  
    ) RETURN NUMBER;
```

Description

Returns the measure value of the next shape point on a geometric segment after a specified measure value or LRS point.

Parameters

geom_segment

Geometric segment.

measure

Measure value on the geometric segment for which to return the measure value of the next shape point.

point

Point for which to return the measure value of the next shape point. If `point` is not on `geom_segment`, the point on the geometric segment closest to the specified point is computed, and the measure value of the next shape point after that point is returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

If `measure` or `point` identifies the end point of the geometric segment, a null value is returned.

An exception is raised if `measure` is not a valid value for `geom_segment` or if `point` is not a valid LRS point.

Contrast this function with [SDO_LRS.GET_PREV_SHAPE_PT_MEASURE](#), which returns the measure value of the previous shape point on a geometric segment before a specified measure value or LRS point.

The `_3D` format of this function ([SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE_3D](#)) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the measure value of the next shape point after measure 14 on the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE(a.route_geometry, 14)
       FROM lrs_routes a WHERE a.route_id = 1;
```

```
SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE(A.ROUTE_GEOMETRY,14)
```

```
-----
```

18

SDO_LRS.GET_PREV_SHAPE_PT

Format

```
SDO_LRS.GET_PREV_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    measure      IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.GET_PREV_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    measure      IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.GET_PREV_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    point        IN SDO_GEOMETRY  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.GET_PREV_SHAPE_PT(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    point        IN SDO_GEOMETRY  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns the previous shape point on a geometric segment before a specified measure value or LRS point.

Parameters

geom_segment

Geometric segment.

measure

Measure value on the geometric segment for which to return the previous shape point.

point

Point for which to return the previous shape point. If `point` is not on `geom_segment`, the point on the geometric segment closest to the specified point is computed, and the closest shape point before that point is returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

If `measure` or `point` identifies the start point of the geometric segment, a null value is returned.

An exception is raised if `measure` is not a valid value for `geom_segment` or if `point` is not a valid LRS point.

Contrast this function with [SDO_LRS.GET_NEXT_SHAPE_PT](#), which returns the next shape point on a geometric segment after a specified measure value or LRS point.

The `_3D` format of this function (`SDO_LRS.GET_PREV_SHAPE_PT_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the closest shape point to measure 14 and before measure 14 on the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.GET_PREV_SHAPE_PT(a.route_geometry, 14)
       FROM lrs_routes a WHERE a.route_id = 1;
```

```
SDO_LRS.GET_PREV_SHAPE_PT(A.ROUTE_GEOMETRY,14) (SDO_GTYPE, SDO_SRID, SDO_POINT(X,
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
12, 4, 12))
```

SDO_LRS.GET_PREV_SHAPE_PT_MEASURE

Format

```
SDO_LRS.GET_PREV_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    measure      IN NUMBER  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.GET_PREV_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    measure      IN NUMBER  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.GET_PREV_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    point        IN SDO_GEOMETRY  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.GET_PREV_SHAPE_PT_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    point        IN SDO_GEOMETRY  
    ) RETURN NUMBER;
```

Description

Returns the measure value of the previous shape point on a geometric segment before a specified measure value or LRS point.

Parameters

geom_segment

Geometric segment.

measure

Measure value on the geometric segment for which to return the measure value of the previous shape point.

point

Point for which to return the measure value of the previous shape point. If `point` is not on `geom_segment`, the point on the geometric segment closest to the specified point is computed, and the measure value of the closest shape point before that point is returned.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

If `measure` or `point` identifies the start point of the geometric segment, a null value is returned.

An exception is raised if `measure` is not a valid value for `geom_segment` or if `point` is not a valid LRS point.

Contrast this function with [SDO_LRS.GET_NEXT_SHAPE_PT_MEASURE](#), which returns the measure value of the next shape point on a geometric segment after a specified measure value or LRS point.

The `_3D` format of this function ([SDO_LRS.GET_PREV_SHAPE_PT_MEASURE_3D](#)) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the measure value of the closest shape point to measure 14 and before measure 14 on the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.GET_PREV_SHAPE_PT_MEASURE(a.route_geometry, 14)
       FROM lrs_routes a WHERE a.route_id = 1;
```

```
SDO_LRS.GET_PREV_SHAPE_PT_MEASURE(A.ROUTE_GEOMETRY,14)
```

12

SDO_LRS.IS_GEOM_SEGMENT_DEFINED

Format

```
SDO_LRS.IS_GEOM_SEGMENT_DEFINED(
  geom_segment IN SDO_GEOMETRY
  [, dim_array  IN SDO_DIM_ARRAY]
) RETURN VARCHAR2;
```

Description

Checks if an LRS segment is defined correctly.

Parameters

geom_segment

Geometric segment to be checked.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns TRUE if `geom_segment` is defined correctly and FALSE if `geom_segment` is not defined correctly.

The start and end measures of `geom_segment` must be defined (cannot be null), and any measures assigned must be in an ascending or descending order along the segment direction.

The `_3D` format of this function (`SDO_LRS.IS_GEOM_SEGMENT_DEFINED_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

See also the [SDO_LRS.VALID_GEOM_SEGMENT](#) function.

Examples

The following example checks if the geometric segment representing Route 1 is defined. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.IS_GEOM_SEGMENT_DEFINED(route_geometry)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.IS_GEOM_SEGMENT_DEFINED(ROUTE_GEOMETRY)
```

```
-----
TRUE
```

SDO_LRS.IS_MEASURE_DECREASING

Format

```
SDO_LRS.IS_MEASURE_DECREASING(
    geom_segment IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
) RETURN VARCHAR2;
```

Description

Checks if the measure values along an LRS segment are decreasing (that is, descending in numerical value).

Parameters

geom_segment

Geometric segment to be checked.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns TRUE if the measure values along an LRS segment are decreasing and FALSE if the measure values along an LRS segment are not decreasing.

The start and end measures of `geom_segment` must be defined (cannot be null).

The `_3D` format of this function (`SDO_LRS.IS_MEASURE_DECREASING_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

See also the [SDO_LRS.IS_MEASURE_INCREASING](#) function.

Examples

The following example checks if the measure values along the geometric segment representing Route 1 are decreasing. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.IS_MEASURE_DECREASING(a.route_geometry, m.diminfo)
   FROM lrs_routes a, user_sdo_geom_metadata m
   WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
   AND a.route_id = 1;
```

```
SDO_LRS.IS_MEASURE_DECREASING(A.ROUTE_GEOMETRY,M.DIMINFO)
```

```
-----
FALSE
```

SDO_LRS.IS_MEASURE_INCREASING

Format

```
SDO_LRS.IS_MEASURE_INCREASING(  
    geom_segment IN SDO_GEOMETRY  
    [, dim_array  IN SDO_DIM_ARRAY]  
    ) RETURN VARCHAR2;
```

Description

Checks if the measure values along an LRS segment are increasing (that is, ascending in numerical value).

Parameters

geom_segment

Geometric segment to be checked.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns TRUE if the measure values along an LRS segment are increasing and FALSE if the measure values along an LRS segment are not increasing.

The start and end measures of `geom_segment` must be defined (cannot be null).

The `_3D` format of this function (`SDO_LRS.IS_MEASURE_INCREASING_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

See also the [SDO_LRS.IS_MEASURE_DECREASING](#) function.

Examples

The following example checks if the measure values along the geometric segment representing Route 1 are increasing. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.IS_MEASURE_INCREASING(a.route_geometry, m.diminfo)  
    FROM lrs_routes a, user_sdo_geom_metadata m  
    WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'  
    AND a.route_id = 1;
```

```
SDO_LRS.IS_MEASURE_INCREASING(A.ROUTE_GEOMETRY,M.DIMINFO)
```

```
-----  
TRUE
```

SDO_LRS.IS_SHAPE_PT_MEASURE

Format

```
SDO_LRS.IS_SHAPE_PT_MEASURE(
    geom_segment IN SDO_GEOMETRY,
    measure      IN NUMBER
) RETURN VARCHAR2;
```

or

```
SDO_LRS.IS_SHAPE_PT_MEASURE(
    geom_segment IN SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    measure      IN NUMBER
) RETURN VARCHAR2;
```

Description

Checks if a specified measure value is associated with a shape point on a geometric segment.

Parameters

geom_segment

Geometric segment to be checked.

measure

Measure value on the geometric segment to check if it is a shape point.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns TRUE if the specified measure value is associated with a shape point and FALSE if the measure value is not associated with a shape point.

An exception is raised if `measure` is not a valid value for `geom_segment`.

The `_3D` format of this function (`SDO_LRS.IS_SHAPE_PT_MEASURE_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example checks if measure 14 on the geometric segment representing Route 1 is a shape point. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.IS_SHAPE_PT_MEASURE(a.route_geometry, 14)
       FROM lrs_routes a WHERE a.route_id = 1;
```

```
SDO_LRS.IS_SHAPE_PT_MEASURE(A.ROUTE_GEOMETRY,14)
```

FALSE

SDO_LRS.LOCATE_PT

Format

```
SDO_LRS.LOCATE_PT(
    geom_segment IN SDO_GEOMETRY,
    measure      IN NUMBER
    [, offset    IN NUMBER
    ]
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.LOCATE_PT(
    geom_segment IN SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    measure      IN NUMBER
    [, offset    IN NUMBER]
) RETURN SDO_GEOMETRY;
```

Description

Returns the point located at a specified distance from the start of a geometric segment.

Parameters

geom_segment

Geometric segment to be checked to see if it falls within the measure range of `measure`.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

measure

Distance to measure from the start point of `geom_segment`.

offset

Distance to measure perpendicularly from the point that is located at `measure` units from the start point of `geom_segment`. The default is 0 (that is, the point is on `geom_segment`).

Usage Notes

This function returns the referenced point. For example, on a highway, the point might represent the location of an accident.

The unit of measurement for `offset` is the same as for the coordinate system associated with `geom_segment`. For geodetic data, the default unit of measurement is meters.

With geodetic data using the WGS 84 coordinate system, this function can be used to return the longitude and latitude coordinates of any point on or offset from the segment.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality, or if the location is out of range.

The `_3D` format of this function (`SDO_LRS.LOCATE_PT_3D`) is available; however, the `offset` parameter is not available for `SDO_LRS.LOCATE_PT_3D`. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

For more information about locating a point on a geometric segment, see [Section 7.5.8](#).

Examples

The following example creates a table for automobile accident data, inserts a record for an accident at the point at measure 9 and on (that is, offset 0) the geometric segment representing Route 1, and displays the data. (The accident table is deliberately oversimplified. This example also uses the route definition from the example in [Section 7.7](#).)

```
-- Create a table for accidents.
CREATE TABLE accidents (
  accident_id NUMBER PRIMARY KEY,
  route_id NUMBER,
  accident_geometry SDO_GEOMETRY);

-- Insert an accident record.
DECLARE
geom_segment SDO_GEOMETRY;

BEGIN

SELECT SDO_LRS.LOCATE_PT(a.route_geometry, 9, 0) into geom_segment
  FROM lrs_routes a WHERE a.route_name = 'Route1';

INSERT INTO accidents VALUES(1, 1, geom_segment);

END;
/

SELECT * from accidents;

ACCIDENT_ID    ROUTE_ID
-----
ACCIDENT_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_OR
-----
                1          1
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
9, 4, 9))
```

SDO_LRS.LRS_INTERSECTION

Format

```
SDO_LRS.LRS_INTERSECTION(
    geom_1    IN SDO_GEOMETRY,
    dim_array_1 IN SDO_DIM_ARRAY,
    geom_2    IN SDO_GEOMETRY,
    dim_array_2 IN SDO_DIM_ARRAY
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.LRS_INTERSECTION(
    geom_1 IN SDO_GEOMETRY,
    geom_2 IN SDO_GEOMETRY,
    tolerance IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns an LRS geometry object that is the topological intersection (AND operation) of two geometry objects where one or both are LRS geometries.

Parameters

geom_1

Geometry object.

dim_array_1

Dimensional information array corresponding to `geom_1`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

geom_2

Geometry object.

dim_array_2

Dimensional information array corresponding to `geom_2`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

tolerance

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

Note: This function is new with Oracle Spatial release 10.2.0.3.

This function performs essentially the same intersection operation as the [SDO_GEOM.SDO_INTERSECTION](#) function (described in [Chapter 15](#)), except that `SDO_LRS.LRS_INTERSECTION` is designed to return a valid LRS geometry (point, line

string, or multiline string) where one or both of the geometry-related input parameters are LRS geometries. (If neither input geometry is an LRS geometry, this function operates the same as the [SDO_GEOM.SDO_INTERSECTION](#) function.)

The returned geometry is an LRS line string, multiline string, or point geometry that includes measure dimension information. The measure values reflect those in the first LRS geometry specified as an input parameter.

The first LRS geometry specified as an input parameter must not be a polygon; it must be a line string, multiline string, or point.

If an LRS line string (geometric segment) intersects a line string (LRS or standard), the result is an LRS point; if an LRS line string intersects a polygon, the result is an LRS line string.

If the function format with `tolerance` is used, all geometry objects must be defined using 4-digit `SDO_GTYPE` values (explained in [Section 2.2.1](#)).

An exception is raised if `geom_1` and `geom_2` are based on different coordinate systems.

Examples

The following example shows an LRS geometric segment (illustrated in [Figure 7–20](#) in [Section 7.7](#)) intersected by a vertical line from (8,2) to (8,6). The result is an LRS point geometry, in which the measure value (8) reflects the measure for that point (designated as Exit 3 in [Figure 7–20](#)) in the `geom_1` geometry. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- Intersection of LRS segment and standard line segment
SELECT SDO_LRS.LRS_INTERSECTION(route_geometry,
    SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,2,1),
        SDO_ORDINATE_ARRAY(8,2, 8,6)), 0.005)
FROM lrs_routes WHERE route_id = 1;

SDO_LRS.LRS_INTERSECTION(ROUTE_GEOMETRY,SDO_GEOMETRY(2002,NULL,NULL,SDO_ELEM_INF
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
8, 4, 8))
```

The following example shows an LRS geometric segment (illustrated in [Figure 7–20](#) in [Section 7.7](#)) intersected by a vertical line from (12,2) to (12,6). The result is an LRS line string geometry, in which the measure values (12 and 14) reflect measures for points (the first of which is designated as Exit 4 in [Figure 7–20](#)) in the `geom_1` geometry. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.LRS_INTERSECTION(route_geometry,
    SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,2,1),
        SDO_ORDINATE_ARRAY(12,2, 12,6)), 0.005)
FROM lrs_routes WHERE route_id = 1;

SDO_LRS.LRS_INTERSECTION(ROUTE_GEOMETRY,SDO_GEOMETRY(2002,NULL,NULL,SDO_ELEM_INF
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
12, 4, 12, 12, 6, 14))
```

SDO_LRS.MEASURE_RANGE

Format

```
SDO_LRS.MEASURE_RANGE(
    geom_segment IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
) RETURN NUMBER;
```

Description

Returns the measure range of a geometric segment, that is, the difference between the start measure and end measure.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function subtracts the start measure of `geom_segment` from the end measure of `geom_segment`.

The `_3D` format of this function (`SDO_LRS.MEASURE_RANGE_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example returns the measure range of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.MEASURE_RANGE(route_geometry)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.MEASURE_RANGE(ROUTE_GEOMETRY)
-----
```

27

SDO_LRS.MEASURE_TO_PERCENTAGE

Format

```
SDO_LRS.MEASURE_TO_PERCENTAGE(  
    geom_segment IN SDO_GEOMETRY,  
    measure      IN NUMBER  
    ) RETURN NUMBER;
```

or

```
SDO_LRS.MEASURE_TO_PERCENTAGE(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    measure      IN NUMBER  
    ) RETURN NUMBER;
```

Description

Returns the percentage (0 to 100) that a specified measure is of the measure range of a geometric segment.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

measure

Measure value. This function returns the percentage that this measure value is of the measure range.

Usage Notes

This function returns a number (0 to 100) that is the percentage of the measure range that the specified measure represents. (The measure range is the end measure minus the start measure.) For example, if the measure range of `geom_segment` is 50 and `measure` is 20, the function returns 40 (because $20/50 = 40\%$).

This function performs the reverse of the [SDO_LRS.PERCENTAGE_TO_MEASURE](#) function, which returns the measure that corresponds to a percentage value.

An exception is raised if `geom_segment` or `measure` is invalid.

Examples

The following example returns the percentage that 5 is of the measure range of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).) The measure range of this segment is 27, and 5 is approximately 18.5 percent of 27.


```
SELECT SDO_LRS.MEASURE_TO_PERCENTAGE(a.route_geometry, m.diminfo, 5)
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;
```

```
SDO_LRS.MEASURE_TO_PERCENTAGE(A.ROUTE_GEOMETRY,M.DIMINFO,5)
-----
18.5185185
```

SDO_LRS.OFFSET_GEOM_SEGMENT

Format

```
SDO_LRS.OFFSET_GEOM_SEGMENT(  
    geom_segment IN SDO_GEOMETRY,  
    start_measure IN NUMBER,  
    end_measure  IN NUMBER,  
    offset       IN NUMBER,  
    tolerance    IN NUMBER DEFAULT 1.0e-8  
    [, unit      IN VARCHAR2]  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.OFFSET_GEOM_SEGMENT(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    start_measure IN NUMBER,  
    end_measure  IN NUMBER,  
    offset       IN NUMBER  
    [, unit      IN VARCHAR2]  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns the geometric segment at a specified offset from a geometric segment.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

start_measure

Start measure of `geom_segment` at which to start the offset operation.

end_measure

End measure of `geom_segment` at which to start the offset operation.

offset

Distance to measure perpendicularly from the points along `geom_segment`. Positive offset values are to the left of `geom_segment`; negative offset values are to the right of `geom_segment`.

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

unit

Unit of measurement specification: a quoted string with one or both of the following keywords:

- `unit` and an `SDO_UNIT` value from the `MDSYS.SDO_DIST_UNITS` table. See [Section 2.8](#) for more information about unit of measurement specification.
- `arc_tolerance` and an arc tolerance value. See the Usage Notes for the `SDO_GEOM.SDO_ARC_DENSIFY` function in [Chapter 15](#) for more information about the `arc_tolerance` keyword.

For example: `'unit=km arc_tolerance=0.05'`

If the input geometry is geodetic data, this parameter is required, and `arc_tolerance` must be specified. If the input geometry is Cartesian or projected data, `arc_tolerance` has no effect and should not be specified.

If this parameter is not specified for a Cartesian or projected geometry, or if the `arc_tolerance` keyword is specified for a geodetic geometry but the `unit` keyword is not specified, the unit of measurement associated with the data is assumed.

Usage Notes

`start_measure` and `end_measure` can be any points on the geometric segment. They do not have to be in any specific order. For example, `start_measure` and `end_measure` can be 5 and 10, respectively, or 10 and 5, respectively.

The direction and measures of the resulting geometric segment are preserved (that is, they reflect the original segment).

The geometry type of `geom_segment` must be line or multiline. For example, it cannot be a polygon.

An exception is raised if `geom_segment`, `start_measure`, or `end_measure` is invalid.

Examples

The following example returns the geometric segment 2 distance units to the left (positive offset 2) of the segment from measures 5 through 10 of Route 1. Note in `SDO_ORDINATE_ARRAY` of the returned segment that the Y values (6) are 2 greater than the Y values (4) of the relevant part of the original segment. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- Create a segment offset 2 to the left from measures 5 through 10.
-- First, display the original segment; then, offset.
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))

SELECT SDO_LRS.OFFSET_GEOM_SEGMENT(a.route_geometry, m.diminfo, 5, 10, 2)
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;

SDO_LRS.OFFSET_GEOM_SEGMENT(A.ROUTE_GEOMETRY,M.DIMINFO,5,10,2) (SDO_GTYPE, SDO_SR
```

```
-----  
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(  
5, 6, 5, 10, 6, 10))
```

SDO_LRS.PERCENTAGE_TO_MEASURE

Format

```
SDO_LRS.PERCENTAGE_TO_MEASURE(
    geom_segment IN SDO_GEOMETRY,
    percentage    IN NUMBER
) RETURN NUMBER;
```

or

```
SDO_LRS.PERCENTAGE_TO_MEASURE(
    geom_segment IN SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    percentage    IN NUMBER
) RETURN NUMBER;
```

Description

Returns the measure value of a specified percentage (0 to 100) of the measure range of a geometric segment.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

percentage

Percentage value. Must be from 0 to 100. This function returns the measure value corresponding to this percentage of the measure range.

Usage Notes

This function returns the measure value corresponding to the specified percentage of the measure range. (The measure range is the end measure minus the start measure.) For example, if the measure range of `geom_segment` is 50 and `percentage` is 40, the function returns 20 (because 40% of 50 = 20).

This function performs the reverse of the [SDO_LRS.MEASURE_TO_PERCENTAGE](#) function, which returns the percentage value that corresponds to a measure.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality, or if `percentage` is less than 0 or greater than 100.

Examples

The following example returns the measure that is 50 percent of the measure range of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).) The measure range of this segment is 27, and 50 percent of 27 is 13.5.

```
SELECT SDO_LRS.PERCENTAGE_TO_MEASURE(a.route_geometry, m.diminfo, 50)
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;
```

```
SDO_LRS.PERCENTAGE_TO_MEASURE(A.ROUTE_GEOMETRY,M.DIMINFO,50)
```

```
-----  
13.5
```

SDO_LRS.PROJECT_PT

Format

```
SDO_LRS.PROJECT_PT(
    geom_segment IN SDO_GEOMETRY,
    point        IN SDO_GEOMETRY,
    tolerance    IN NUMBER DEFAULT 1.0e-8
    [, offset    OUT NUMBER]
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.PROJECT_PT(
    geom_segment IN SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    point        IN SDO_GEOMETRY
    [, point_dim_array IN SDO_DIM_ARRAY]
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.PROJECT_PT(
    geom_segment IN SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    point        IN SDO_GEOMETRY,
    point_dim_array IN SDO_DIM_ARRAY
    [, offset    OUT NUMBER]
) RETURN SDO_GEOMETRY;
```

Description

Returns the projection point of a specified point. The projection point is on the geometric segment.

Parameters

geom_segment

Geometric segment to be checked.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

point

Point to be projected.

tolerance

Tolerance value (see [Section 1.5.5](#) and [Section 7.6](#)). The default value is 0.00000001.

point_dim_array

Dimensional information array corresponding to `point`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

offset

Offset (shortest distance) from the point to the geometric segment.

Usage Notes

This function returns the projection point (including its measure) of a specified point (`point`). The projection point is on the geometric segment.

If multiple projection points exist, the first projection point encountered from the start point is returned.

If you specify the output parameter `offset`, the function stores the signed offset (shortest distance) from the point to the geometric segment. For more information about the offset to a geometric segment, see [Section 7.1.5](#).

An exception is raised if `geom_segment` or `point` has an invalid geometry type or dimensionality, or if `geom_segment` and `point` are based on different coordinate systems.

The `_3D` format of this function (`SDO_LRS.PROJECT_PT_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

For more information about projecting a point onto a geometric segment, see [Section 7.5.9](#).

Examples

The following example returns the point (9,4,9) on the geometric segment representing Route 1 that is closest to the specified point (9,3,NULL). (This example uses the definitions from the example in [Section 7.7](#).)

```
-- Point 9,3,NULL is off the road; should return 9,4,9.
SELECT  SDO_LRS.PROJECT_PT(route_geometry,
      SDO_GEOMETRY(3301, NULL, NULL,
        SDO_ELEM_INFO_ARRAY(1, 1, 1),
        SDO_ORDINATE_ARRAY(9, 3, NULL)) )
FROM    lrs_routes WHERE route_id = 1;

SDO_LRS.PROJECT_PT(ROUTE_GEOMETRY,SDO_GEOMETRY(3301,NULL,NULL,SDO_EL
-----
SDO_GEOMETRY(3301, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
9, 4, 9))
```

SDO_LRS.REDEFINE_GEOM_SEGMENT

Format

```
SDO_LRS.REDEFINE_GEOM_SEGMENT(
  geom_segment IN OUT SDO_GEOMETRY
  [, start_measure IN NUMBER,
  end_measure IN NUMBER]);
```

or

```
SDO_LRS.REDEFINE_GEOM_SEGMENT(
  geom_segment IN OUT SDO_GEOMETRY,
  dim_array IN SDO_DIM_ARRAY
  [, start_measure IN NUMBER,
  end_measure IN NUMBER]);
```

Description

Populates the measures of all shape points based on the start and end measures of a geometric segment, overriding any previously assigned measures between the start point and end point.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

start_measure

Distance measured from the start point of a geometric segment to the start point of the linear feature. The default is the existing value (if any) in the measure dimension; otherwise, the default is 0.

end_measure

Distance measured from the end point of a geometric segment to the start point of the linear feature. The default is the existing value (if any) in the measure dimension; otherwise, the default is the cartographic length of the segment.

Usage Notes

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality, or if `start_measure` or `end_measure` is out of range.

The *_3D* format of this procedure (`SDO_LRS.REDEFINE_GEOM_SEGMENT_3D`) is available. For information about *_3D* formats of LRS functions and procedures, see [Section 7.4](#).

For more information about redefining a geometric segment, see [Section 7.5.2](#).

Examples

The following example redefines a geometric segment, effectively converting miles to kilometers in the measure values. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- First, display the original segment; then, redefine.
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))

-- Redefine geometric segment to "convert" miles to kilometers.
DECLARE
geom_segment SDO_GEOMETRY;
dim_array SDO_DIM_ARRAY;

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
  WHERE a.route_name = 'Route1';
SELECT m.diminfo into dim_array from
  user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

-- "Convert" mile measures to kilometers (27 * 1.609 = 43.443).
SDO_LRS.REDEFINE_GEOM_SEGMENT (geom_segment,
  dim_array,
  0,      -- Zero starting measure: LRS segment starts at start of route.
  43.443); -- End of LRS segment. 27 miles = 43.443 kilometers.

-- Update and insert geometries into table, to display later.
UPDATE lrs_routes a SET a.route_geometry = geom_segment
  WHERE a.route_id = 1;

END;
/

PL/SQL procedure successfully completed.

-- Display the redefined segment, with all measures "converted."
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 3.218, 8, 4, 12.872, 12, 4, 19.308, 12, 10, 28.962, 8, 10, 35.398
, 5, 14, 43.443))
```

SDO_LRS.RESET_MEASURE

Format

```
SDO_LRS.RESET_MEASURE(
    geom_segment IN OUT SDO_GEOMETRY
    [, dim_array IN SDO_DIM_ARRAY]);
```

Description

Sets all measures of a geometric segment, including the start and end measures, to null values, overriding any previously assigned measures.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

Examples

The following example sets all measures of a geometric segment to null values. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- First, display the original segment; then, redefine.
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))

-- Reset geometric segment measures.
DECLARE
geom_segment SDO_GEOMETRY;

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
    WHERE a.route_name = 'Route1';

SDO_LRS.RESET_MEASURE (geom_segment);

-- Update and insert geometries into table, to display later.
UPDATE lrs_routes a SET a.route_geometry = geom_segment
    WHERE a.route_id = 1;

END;
/
```

PL/SQL procedure successfully completed.

-- Display the segment, with all measures set to null.

```
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;
```

```
ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
```

```
-----  
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(  
2, 2, NULL, 2, 4, NULL, 8, 4, NULL, 12, 4, NULL, 12, 10, NULL, 8, 10, NULL, 5, 1  
4, NULL))
```

SDO_LRS.REVERSE_GEOMETRY

Format

```
SDO_LRS.REVERSE_GEOMETRY(
    geom          IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
) RETURN SDO_GEOMETRY;
```

Description

Returns a new geometric segment by reversing the measure values and the direction of the original geometric segment.

Parameters

geom

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function:

- Reverses the measure values of `geom`

That is, the start measure of `geom` is the end measure of the returned geometric segment, the end measure of `geom` is the start measure of the returned geometric segment, and all other measures are adjusted accordingly.
- Reverses the direction of `geom`

Compare this function with [SDO_LRS.REVERSE_MEASURE](#), which reverses only the measure values (not the direction) of a geometric segment.

To reverse the vertices of a non-LRS line string geometry, use the [SDO_UTIL.REVERSE_LINestring](#) function, which is described in [Chapter 20](#).

An exception is raised if `geom` has an invalid geometry type or dimensionality. The geometry type must be a line or multiline, and the dimensionality must be 3 (two dimensions plus the measure dimension).

The `_3D` format of this function (`SDO_LRS.REVERSE_GEOMETRY_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example reverses the measure values and the direction of the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- Reverse direction and measures (for example, to prepare for
-- concatenating with another road).
-- First, display the original segment; then, reverse.
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;
```

```
ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATE_ARRAY(
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))
```

```
SELECT SDO_LRS.REVERSE_GEOMETRY(a.route_geometry, m.diminfo)
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;
```

```
SDO_LRS.REVERSE_GEOMETRY(A.ROUTE_GEOMETRY,M.DIMINFO) (SDO_GTYPE, SDO_SRID, SDO_PO
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
5, 14, 27, 8, 10, 22, 12, 10, 18, 12, 4, 12, 8, 4, 8, 2, 4, 2, 2, 2, 0))
```

Note in the returned segment that the M values (measures) now go in descending order from 27 to 0, and the segment start and end points have the opposite X and Y values as in the original segment (5,14 and 2,2 here, as opposed to 2,2 and 5,14 in the original).

SDO_LRS.REVERSE_MEASURE

Format

```
SDO_LRS.REVERSE_MEASURE(
    geom_segment IN SDO_GEOMETRY
    [, dim_array  IN SDO_DIM_ARRAY]
) RETURN SDO_GEOMETRY;
```

Description

Returns a new geometric segment by reversing the measure values, but not the direction, of the original geometric segment.

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function:

- Reverses the measure values of `geom_segment`

That is, the start measure of `geom_segment` is the end measure of the returned geometric segment, the end measure of `geom_segment` is the start measure of the returned geometric segment, and all other measures are adjusted accordingly.
- Does not affect the direction of `geom_segment`

Compare this function with [SDO_LRS.REVERSE_GEOMETRY](#), which reverses both the direction and the measure values of a geometric segment.

An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

The `_3D` format of this function (`SDO_LRS.REVERSE_MEASURE_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Note: The behavior of the `SDO_LRS.REVERSE_MEASURE` function changed after release 8.1.7. In release 8.1.7, `REVERSE_MEASURE` reversed both the measures and the segment direction. However, if you want to have this same behavior with subsequent releases, you must use the [SDO_LRS.REVERSE_GEOMETRY](#) function.

Examples

The following example reverses the measure values of the geometric segment representing Route 1, but does not affect the direction. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- First, display the original segment; then, reverse.
SELECT a.route_geometry FROM lrs_routes a WHERE a.route_id = 1;

ROUTE_GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDIN
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 2, 4, 2, 8, 4, 8, 12, 4, 12, 12, 10, 18, 8, 10, 22, 5, 14, 27))
```

```
SELECT SDO_LRS.REVERSE_MEASURE(a.route_geometry, m.diminfo)
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;
```

```
SDO_LRS.REVERSE_MEASURE(A.ROUTE_GEOMETRY,M.DIMINFO) (SDO_GTYPE, SDO_SRID, SDO_POI
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 27, 2, 4, 25, 8, 4, 19, 12, 4, 15, 12, 10, 9, 8, 10, 5, 5, 14, 0))
```

Note in the returned segment that the M values (measures) now go in descending order from 27 to 0, but the segment start and end points have the same X and Y values as in the original segment (2,2 and 5,14).

SDO_LRS.SET_PT_MEASURE

Format

```
SDO_LRS.SET_PT_MEASURE(
    geom_segment IN OUT SDO_GEOMETRY,
    point        IN SDO_GEOMETRY,
    measure      IN NUMBER) RETURN VARCHAR2;
```

or

```
SDO_LRS.SET_PT_MEASURE(
    geom_segment IN OUT SDO_GEOMETRY,
    dim_array    IN SDO_DIM_ARRAY,
    point        IN SDO_GEOMETRY,
    pt_dim_array IN SDO_DIM_ARRAY,
    measure      IN NUMBER) RETURN VARCHAR2;
```

or

```
SDO_LRS.SET_PT_MEASURE(
    point IN OUT SDO_GEOMETRY,
    measure IN NUMBER) RETURN VARCHAR2;
```

or

```
SDO_LRS.SET_PT_MEASURE(
    point IN OUT SDO_GEOMETRY,
    dim_array IN SDO_DIM_ARRAY,
    measure IN NUMBER) RETURN VARCHAR2;
```

Description

Sets the measure value of a specified point.

Parameters

geom_segment

Geometric segment containing the point.

dim_array

Dimensional information array corresponding to `geom_segment` (in the second format) or `point` (in the fourth format), usually selected from one of the xxx_SDO_GEOM_METADATA views (described in [Section 2.6](#)).

point

Point for which the measure value is to be set.

pt_dim_array

Dimensional information array corresponding to point (in the second format), usually selected from one of the xxx_SDO_GEOM_METADATA views (described in [Section 2.6](#)).

measure

Measure value to be assigned to the specified point.

Usage Notes

The function returns TRUE if the measure value was successfully set, and FALSE if the measure value was not set.

If both `geom_segment` and `point` are specified, the behavior of the procedure depends on whether or not `point` is a shape point on `geom_segment`:

- If `point` is a shape point on `geom_segment`, the measure value of `point` is set.
- If `point` is not a shape point on `geom_segment`, the shape point on `geom_segment` that is nearest to `point` is found, and the measure value of that shape point is set.

The `_3D` format of this function (`SDO_LRS.SET_PT_MEASURE_3D`) is available; however, only the formats that include the `geom_segment` parameter are available for `SDO_LRS.SET_PT_MEASURE_3D`. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

An exception is raised if `geom_segment` or `point` is invalid.

Examples

The following example sets the measure value of point (8,10) to 20. (This example uses the definitions from the example in [Section 7.7](#).)

```
-- Set the measure value of point 8,10 to 20 (originally 22).
DECLARE
geom_segment SDO_GEOMETRY;
dim_array SDO_DIM_ARRAY;
result VARCHAR2(32);

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
WHERE a.route_name = 'Route1';
SELECT m.diminfo into dim_array from
user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

-- Set the measure value of point 8,10 to 20 (originally 22).
result := SDO_LRS.SET_PT_MEASURE (geom_segment,
SDO_GEOMETRY(3301, NULL, NULL,
SDO_ELEM_INFO_ARRAY(1, 1, 1),
SDO_ORDINATE_ARRAY(8, 10, 22)),
20);

-- Display the result.
DBMS_OUTPUT.PUT_LINE('Returned value = ' || result);

END;
/
Returned value = TRUE
```

PL/SQL procedure successfully completed.

SDO_LRS.SPLIT_GEOM_SEGMENT

Format

```
SDO_LRS.SPLIT_GEOM_SEGMENT(  
    geom_segment IN SDO_GEOMETRY,  
    split_measure IN NUMBER,  
    segment_1    OUT SDO_GEOMETRY,  
    segment_2    OUT SDO_GEOMETRY);
```

or

```
SDO_LRS.SPLIT_GEOM_SEGMENT(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    split_measure IN NUMBER,  
    segment_1    OUT SDO_GEOMETRY,  
    segment_2    OUT SDO_GEOMETRY);
```

Description

Splits a geometric segment into two geometric segments.

Parameters

geom_segment

Geometric segment to be split.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

split_measure

Distance measured from the start point of a geometric segment to the split point.

segment_1

First geometric segment: from the start point of `geom_segment` to the split point.

segment_2

Second geometric segment: from the split point to the end point of `geom_segment`.

Usage Notes

An exception is raised if `geom_segment` or `split_measure` is invalid.

The directions and measures of the resulting geometric segments are preserved.

The *_3D* format of this procedure (`SDO_LRS.SPLIT_GEOM_SEGMENT_3D`) is available. For information about *_3D* formats of LRS functions and procedures, see [Section 7.4](#).

For more information about splitting a geometric segment, see [Section 7.5.4](#).

Examples

The following example defines the geometric segment, splits it into two segments, then concatenates those segments. (This example uses the definitions from the example in [Section 7.7](#). The definitions of `result_geom_1`, `result_geom_2`, and `result_geom_3` are displayed in [Example 7-3](#).)

```

DECLARE
geom_segment SDO_GEOMETRY;
line_string SDO_GEOMETRY;
dim_array SDO_DIM_ARRAY;
result_geom_1 SDO_GEOMETRY;
result_geom_2 SDO_GEOMETRY;
result_geom_3 SDO_GEOMETRY;

BEGIN

SELECT a.route_geometry into geom_segment FROM lrs_routes a
  WHERE a.route_name = 'Route1';
SELECT m.diminfo into dim_array from
  user_sdo_geom_metadata m
  WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY';

-- Define the LRS segment for Route1.
SDO_LRS.DEFINE_GEOM_SEGMENT (geom_segment,
  dim_array,
  0,      -- Zero starting measure: LRS segment starts at start of route.
  27);   -- End of LRS segment is at measure 27.

SELECT a.route_geometry INTO line_string FROM lrs_routes a
  WHERE a.route_name = 'Route1';

-- Split Route1 into two segments.
SDO_LRS.SPLIT_GEOM_SEGMENT(line_string,dim_array,5,result_geom_1,result_geom_2);

-- Concatenate the segments that were just split.
result_geom_3 := SDO_LRS.CONCATENATE_GEOM_SEGMENTS(result_geom_1, dim_array,
result_geom_2, dim_array);

-- Insert geometries into table, to display later.
INSERT INTO lrs_routes VALUES(
  11,
  'result_geom_1',
  result_geom_1
);
INSERT INTO lrs_routes VALUES(
  12,
  'result_geom_2',
  result_geom_2
);
INSERT INTO lrs_routes VALUES(
  13,
  'result_geom_3',
  result_geom_3
);

END;
/

```

SDO_LRS.TRANSLATE_MEASURE

Format

```
SDO_LRS.TRANSLATE_MEASURE(
  geom_segment IN SDO_GEOMETRY,
  translate_m   IN NUMBER
) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.TRANSLATE_MEASURE(
  geom_segment IN SDO_GEOMETRY,
  dim_array    IN SDO_DIM_ARRAY,
  translate_m   IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns a new geometric segment by translating the original geometric segment (that is, shifting the start and end measures by a specified value).

Parameters

geom_segment

Cartographic representation of a linear feature.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

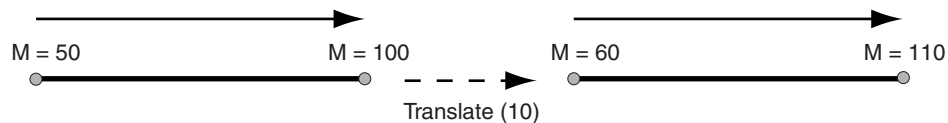
translate_m

Distance measured from the start point of a geometric segment to the start point of the linear feature.

Usage Notes

This function adds `translate_m` to the start and end measures of `geom_segment`. For example, if `geom_segment` has a start measure of 50 and an end measure of 100, and if `translate_m` is 10, the returned geometric segment has a start measure of 60 and an end measure of 110, as shown in [Figure 16–1](#).

Figure 16–1 Translating a Geometric Segment



An exception is raised if `geom_segment` has an invalid geometry type or dimensionality.

The *_3D* format of this function (SDO_LRS.TRANSLATE_MEASURE_3D) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

Examples

The following example translates (shifts) by 10 the geometric segment representing Route 1. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.TRANSLATE_MEASURE(a.route_geometry, m.diminfo, 10)
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;

SDO_LRS.TRANSLATE_MEASURE(A.ROUTE_GEOMETRY,M.DIMINFO,10) (SDO_GTYPE, SDO_SRID, SD
-----
SDO_GEOMETRY(3002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 10, 2, 4, 12, 8, 4, 18, 12, 4, 22, 12, 10, 28, 8, 10, 32, 5, 14, 37))
```

SDO_LRS.VALID_GEOM_SEGMENT

Format

```
SDO_LRS.VALID_GEOM_SEGMENT(  
    geom_segment IN SDO_GEOMETRY  
    [, dim_array  IN SDO_DIM_ARRAY]  
    ) RETURN VARCHAR2;
```

Description

Checks if a geometry object is a valid geometric segment.

Parameters

geom_segment

Geometric segment to be checked for validity.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns TRUE if `geom_segment` is valid and FALSE if `geom_segment` is not valid.

Measure information is assumed to be stored in the last element of the `SDO_DIM_ARRAY` in the Oracle Spatial metadata.

This function only checks for geometry type and number of dimensions of the geometric segment. To further validate measure information, use the [SDO_LRS.IS_GEOM_SEGMENT_DEFINED](#) function.

The `_3D` format of this function (`SDO_LRS.VALID_GEOM_SEGMENT_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example checks if the geometric segment representing Route 1 is valid. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.VALID_GEOM_SEGMENT(route_geometry)  
FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.VALID_GEOM_SEGMENT(ROUTE_GEOMETRY)
```

```
-----  
TRUE
```

SDO_LRS.VALID_LRS_PT

Format

```
SDO_LRS.VALID_LRS_PT(
    point      IN SDO_GEOMETRY
    [, dim_array IN SDO_DIM_ARRAY]
) RETURN VARCHAR2;
```

Description

Checks if an LRS point is valid.

Parameters

point

Point to be checked for validity.

dim_array

Dimensional information array corresponding to `point`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns TRUE if `point` is valid and FALSE if `point` is not valid.

This function checks if `point` is a point with measure information, and it checks for the geometry type and number of dimensions for the point geometry.

All LRS point data must be stored in the `SDO_ELEM_INFO_ARRAY` and `SDO_ORDINATE_ARRAY`, and cannot be stored in the `SDO_POINT` field in the `SDO_GEOMETRY` definition of the point.

The `_3D` format of this function (`SDO_LRS.VALID_LRS_PT_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example checks if point (9,3,NULL) is a valid LRS point. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.VALID_LRS_PT(
    SDO_GEOMETRY(3301, NULL, NULL,
        SDO_ELEM_INFO_ARRAY(1, 1, 1),
        SDO_ORDINATE_ARRAY(9, 3, NULL)),
    m.diminfo)
FROM lrs_routes a, user_sdo_geom_metadata m
WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'
AND a.route_id = 1;

SDO_LRS.VALID_LRS_PT(SDO_GEOMETRY(3301,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,1,1),SDO_
-----
TRUE
```

SDO_LRS.VALID_MEASURE

Format

```
SDO_LRS.VALID_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    measure      IN NUMBER  
    ) RETURN VARCHAR2;
```

or

```
SDO_LRS.VALID_MEASURE(  
    geom_segment IN SDO_GEOMETRY,  
    dim_array    IN SDO_DIM_ARRAY,  
    measure      IN NUMBER  
    ) RETURN VARCHAR2;
```

Description

Checks if a measure falls within the measure range of a geometric segment.

Parameters

geom_segment

Geometric segment to be checked to see if *measure* falls within its measure range.

dim_array

Dimensional information array corresponding to *geom_segment*, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in [Section 2.6](#)).

measure

Measure value to be checked to see if it falls within the measure range of *geom_segment*.

Usage Notes

This function returns TRUE if *measure* falls within the measure range of *geom_segment* and FALSE if *measure* does not fall within the measure range of *geom_segment*.

An exception is raised if *geom_segment* has an invalid geometry type or dimensionality.

The *_3D* format of this function (SDO_LRS.VALID_MEASURE_3D) is available. For information about *_3D* formats of LRS functions, see [Section 7.4](#).

Examples

The following example checks if 50 is a valid measure on the Route 1 segment. The function returns FALSE because the measure range for that segment is 0 to 27. For example, if the route is 27 miles long with mile markers at 1-mile intervals, there is no 50-mile marker because the last marker is the 27-mile marker. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.VALID_MEASURE(route_geometry, 50)
       FROM lrs_routes WHERE route_id = 1;
```

```
SDO_LRS.VALID_MEASURE(ROUTE_GEOMETRY,50)
```

```
-----  
FALSE
```

SDO_LRS.VALIDATE_LRS_GEOMETRY

Format

```
SDO_LRS.VALIDATE_LRS_GEOMETRY(  
    geom_segment IN SDO_GEOMETRY  
    [, dim_array  IN SDO_DIM_ARRAY]  
    ) RETURN VARCHAR2;
```

Description

Checks if an LRS geometry is valid.

Parameters

geom_segment

Geometric segment to be checked.

dim_array

Dimensional information array corresponding to `geom_segment`, usually selected from one of the `xxx_SDO_GEOM_METADATA` views (described in [Section 2.6](#)).

Usage Notes

This function returns TRUE if `geom_segment` is valid and one of the following errors if `geom_segment` is not valid:

- ORA-13331 (invalid LRS segment)
- ORA-13335 (measure information not defined)

The `_3D` format of this function (`SDO_LRS.VALIDATE_LRS_GEOMETRY_3D`) is available. For information about `_3D` formats of LRS functions, see [Section 7.4](#).

Examples

The following example checks if the Route 1 segment is a valid LRS geometry. (This example uses the definitions from the example in [Section 7.7](#).)

```
SELECT SDO_LRS.VALIDATE_LRS_GEOMETRY(a.route_geometry, m.diminfo)  
    FROM lrs_routes a, user_sdo_geom_metadata m  
    WHERE m.table_name = 'LRS_ROUTES' AND m.column_name = 'ROUTE_GEOMETRY'  
    AND a.route_id = 1;
```

```
SDO_LRS.VALIDATE_LRS_GEOMETRY(A.ROUTE_GEOMETRY,M.DIMINFO)
```

```
-----  
TRUE
```

SDO_MIGRATE Package (Upgrading)

The [SDO_MIGRATE.TO_CURRENT](#) subprogram described in this chapter has both procedure and function interfaces. As a procedure, it lets you upgrade spatial geometry tables from previous releases of Spatial; and as a function, it lets you upgrade a single SDO_GEOMETRY object.

[SDO_MIGRATE.TO_CURRENT](#) is the only procedure that you should use for upgrading tables. Do not use the [SDO_MIGRATE.TO_81X](#), [SDO_MIGRATE.FROM_815_TO_81X](#), or [SDO_MIGRATE.TO_734](#) procedures, which were documented in previous Spatial releases but are no longer supported.

SDO_MIGRATE.TO_CURRENT

Format (Any Object-Relational Model Implementation to Current)

```
SDO_MIGRATE.TO_CURRENT(  
    tabname      IN VARCHAR2  
    [, column_name IN VARCHAR2]);
```

or

```
SDO_MIGRATE.TO_CURRENT(  
    tabname      IN VARCHAR2,  
    column_name  IN VARCHAR2  
    [, commit_int IN NUMBER]);
```

Format (Single Object-Relational Model Geometry to Current)

```
SDO_MIGRATE.TO_CURRENT(  
    geom IN SDO_GEOMETRY,  
    dim  IN SDO_DIM_ARRAY  
    ) RETURN SDO_GEOMETRY;
```

Format (Any Relational Model Implementation to Current)

```
SDO_MIGRATE.TO_CURRENT(  
    layer      IN VARCHAR2,  
    newtabname IN VARCHAR2,  
    gidcolumn  IN VARCHAR2,  
    geocolname IN VARCHAR2,  
    layer_gtype IN VARCHAR2,  
    updateflag IN VARCHAR2);
```

Description

Upgrades data from a previous Spatial release to the current release. As a procedure, TO_CURRENT upgrades an entire layer (all geometries in a column); as a function, TO_CURRENT upgrades a single geometry object, which must be of type SDO_GEOMETRY.

For upgrading a layer, the procedure format depends on whether you are upgrading from the Spatial relational model (release 8.1.5 or earlier) or object-relational model (release 8.1.6 or later). See the Usage Notes for the model that applies to you.

You should use this procedure for any spatial layer upgrade. Do not use the SDO_MIGRATE.TO_81X, SDO_MIGRATE.FROM_815_TO_81X, or SDO_MIGRATE.TO_734 procedures, which were documented in previous Spatial releases but are no longer supported.

Parameters

tablename

Table with geometry objects.

column_name

Column in `tablename` that contains geometry objects. If `column_name` is not specified or is specified as null, the column containing geometry objects is upgraded.

commit_int

Number of geometries to upgrade before Spatial performs an internal commit operation. If `commit_int` is not specified, no internal commit operations are performed during the upgrade.

If you specify a `commit_int` value, you can use a smaller rollback segment than would otherwise be needed.

geom

Single geometry object to be upgraded to the current release.

dim

Dimensional information array for the geometry object to be upgraded. The SDO_DIM_ARRAY type is explained in [Section 2.6.3](#).

layer

Name of the layer to be upgraded.

newtablename

Name of the new table to which you are upgrading the data.

gidcolumn

Name of the column in which to store the GID from the old table.

geocolname

Name of the column in the new table where the geometry objects will be inserted.

layer_gtype

One of the following values: POINT or NOTPOINT (default).

If the layer you are upgrading is composed solely of point data, set this parameter to POINT for optimal performance; otherwise, set this parameter to NOTPOINT. If you set the value to POINT and the layer contains any nonpoint geometries, the upgrade might produce invalid data.

updateflag

One of the following values: UPDATE or INSERT (default).

If you are upgrading the layer into an existing populated attribute table, set this parameter to UPDATE; otherwise, set this parameter to INSERT.

Usage Notes for Object-Relational Model Layer and Single Geometry Upgrade

The specified geometry or all geometry objects in the specified layer are upgraded so that their SDO_GTYPE and SDO_ETYPE values are in the format of the current release:

- SDO_GTYPE values of 4 digits are created, using the format (*dltt*) shown in [Table 2-1](#) in [Section 2.2.1](#).
- SDO_ETYPE values are as discussed in [Section 2.2.4](#).

Geometries are ordered so that exterior rings are followed by their interior rings, and coordinates are saved in the correct rotation (counterclockwise for exterior rings, and clockwise for interior rings).

Usage Notes for Relational Model Upgrade

Consider the following when using this procedure:

- The new table must be created before you call this procedure.
- If the data to be upgraded is geodetic, the tolerance value (`SDO_TOLERANCE` column value in the `<layername>_SDODIM` table or view) must be expressed in decimal degrees (for example, 0.00000005).
- The procedure converts geometries from the relational model to the object-relational model.
- A commit operation is performed by this procedure.
- If any of the upgrade steps fails, nothing is upgraded for the layer.
- `layer` is the underlying layer name, without the `_SDOGEOM` suffix.
- The old `SDO_GID` is stored in `gidcolumn`.
- `SDO_GTYPE` values of 4 digits are created, using the format (*dltt*) shown in [Table 2-1](#) in [Section 2.2.1](#).
- `SDO_ETYPE` values are created, using the values discussed in [Section 2.2.4](#).
- The procedure orders geometries so that exterior rings are followed by their interior rings, and it saves coordinates in the correct rotation (counterclockwise for exterior rings, and clockwise for interior rings).

Examples

The following example changes the definitions of geometry objects in the `ROADS` table from the release 8.1.5 or later format to the format of the current release.

```
EXECUTE SDO_MIGRATE.TO_CURRENT('ROADS');
```

SDO_SAM Package (Spatial Analysis and Mining)

The MDSYS.SDO_SAM package contains subprograms for spatial analysis and data mining.

To use the subprograms in this chapter, you must understand the conceptual information about spatial analysis and data mining in [Chapter 8](#).

[Table 18–1](#) lists the spatial analysis and mining subprograms.

Table 18–1 Subprograms for Spatial Analysis and Mining

Function	Description
SDO_SAM.AGGREGATES_FOR_GEOMETRY	Computes the thematic aggregate for a geometry.
SDO_SAM.AGGREGATES_FOR_LAYER	Computes thematic aggregates for a layer of geometries.
SDO_SAM.BIN_GEOMETRY	Computes the most-intersecting tile for a geometry.
SDO_SAM.BIN_LAYER	Assigns each location (and the corresponding row) in a data mining table to a spatial bin.
SDO_SAM.COLOCATED_REFERENCE_FEATURES	Performs a partial predicate-based join of tables, and materializes the join results into a table.
SDO_SAM.SIMPLIFY_GEOMETRY	Simplifies a geometry.
SDO_SAM.SIMPLIFY_LAYER	Simplifies a geometry layer.
SDO_SAM.SPATIAL_CLUSTERS	Computes clusters using the existing R-tree index, and returns a set of SDO_REGION objects where the geometry column specifies the boundary of each cluster and the geometry_key value is set to null.
SDO_SAM.TILED_AGGREGATES	Tiles aggregates for a domain. For each tile, computes the intersecting geometries from the theme table; the values in the aggr_col_string column are weighted proportionally to the area of the intersection, and aggregated according to aggr_col_string.
SDO_SAM.TILED_BINS	Tiles a two-dimensional space and returns geometries corresponding to those tiles.

The rest of this chapter provides reference information on the spatial analysis and mining subprograms, listed in alphabetical order.

SDO_SAM.AGGREGATES_FOR_GEOMETRY

Format

```
SDO_SAM.AGGREGATES_FOR_GEOMETRY(  
    theme_name    IN VARCHAR2,  
    theme_colname IN VARCHAR2,  
    aggr_type_string IN VARCHAR2,  
    aggr_col_string IN VARCHAR2,  
    geom          IN SDO_GEOMETRY,  
    dst_spec      IN VARCHAR2 DEFAULT NULL  
) RETURN NUMBER;
```

Description

Computes the thematic aggregate for a geometry.

Parameters

theme_name

Name of the theme table.

theme_colname

Name of the geometry column in `theme_name`.

aggr_type_string

Any Oracle SQL aggregate function that accepts one or more numeric values and computes a numeric value, such as `SUM`, `MIN`, `MAX`, or `AVG`.

aggr_col_string

Name of a column in `theme_name` on which to compute aggregate values, as explained in the Usage Notes. An example might be a `POPULATION` column.

geom

Geometry object.

dst_spec

A quoted string specifying either a distance buffer or a number of nearest neighbor geometries to consider. See the Usage Notes for an explanation of the format and meaning.

Usage Notes

For a specific geometry, this function identifies the geometries in the `theme_name` table, finds their intersection ratio, multiplies the specified aggregate using this intersection ratio, and aggregates it for the geometry. Specifically, for all rows of the `theme_name` table that intersect with the specified geometry, it returns the value from the following function:

```
aggr_type_string(aggr_col_string * proportional_area_of_intersection(geometry,  
theme_name.theme_colname))
```

The `theme_colname` column must have a spatial index defined on it. For best performance, insert simplified geometries into this column.

The `dst_spec` parameter, if specified, is a quoted string that must contain either of the following:

- The `distance` keyword and optionally the `unit` keyword (unit of measurement associated with the distance value), to specify a buffer around the geometry. For example, `'distance=2 unit=km'` specifies a 2-kilometer buffer around the input geometry. If `dst_spec` is not specified, no buffer is used.

If the `unit` keyword is specified, the value must be an `SDO_UNIT` value from the `MDSYS.SDO_DIST_UNITS` table (for example, `'unit=km'`). If the `unit` keyword is not specified, the unit of measurement associated with the geometry is used. See [Section 2.8](#) for more information about unit of measurement specification.

- The `sdo_num_res` keyword, to specify the number of nearest-neighbor geometries to consider, without considering proportional coverage. For example, `'sdo_num_res=5'` could be used in a query that asks for the populations of the five cities that are nearest to a specified point.

Examples

The following example computes the thematic aggregate for an area with a 3-mile radius around a specified point geometry. In this case, the total population of the area is computed based on the proportion of the circle's area within different counties, assuming uniform distribution of population within the counties.

```
SELECT sdo_sam.agggregates_for_geometry(
  'GEOD_COUNTIES', 'GEOM',
  'sum', 'totpop',
  SDO_GEOMETRY(2001, 8307,
    SDO_POINT_TYPE(-73.943849, 40.6698, NULL),
    NULL, NULL),
  'distance=3 unit=mile')
FROM DUAL a ;
```

SDO_SAM.AGGREGATES_FOR_LAYER

Format

```
SDO_SAM.AGGREGATES_FOR_LAYER(  
    theme_name    IN VARCHAR2,  
    theme_colname IN VARCHAR2,  
    aggr_type_string IN VARCHAR2,  
    aggr_col_string IN VARCHAR2,  
    tablename     IN VARCHAR2,  
    colname       IN VARCHAR2,  
    dst_spec      IN VARCHAR2 DEFAULT NULL  
) RETURN SDO_REGAGGRSET;
```

Description

Computes thematic aggregates for a layer of geometries.

Parameters

theme_name

Name of the theme table.

theme_colname

Name of the geometry column in `theme_name`.

aggr_type_string

Any Oracle SQL aggregate function that accepts one or more numeric values and computes a numeric value, such as `SUM`, `MIN`, `MAX`, or `AVG`.

aggr_col_string

Name of a column in `theme_name` on which to compute aggregate values, as explained in the Usage Notes. An example might be a `POPULATION` column.

tablename

Name of the data mining table.

colname

Name of the column in `tablename` that holds the geometries.

dst_spec

A quoted string specifying either a distance buffer or a number of nearest neighbor geometries to consider. See the Usage Notes for the [SDO_SAM.AGGREGATES_FOR_GEOMETRY](#) function in this chapter for an explanation of the format and meaning.

Usage Notes

For each geometry in `tablename`, this function identifies the geometries in the `theme_name` table, finds their intersection ratio, multiplies the specified aggregate using this intersection ratio, and aggregates it for each geometry in `tablename`. Specifically, for all rows of the `theme_name` table, it returns the value from the following function:

```
aggr_type_string(aggr_col_string * proportional_area_of_intersection(geometry,
theme_name.theme_colname))
```

This function returns an object of type SDO_REGAGGRSET. The SDO_REGAGGRSET object type is defined as:

TABLE OF SDO_REGAGGR

The SDO_REGAGGR object type is defined as:

Name	Null?	Type
REGION_ID		VARCHAR2 (24)
GEOMETRY		MDSYS.SDO_GEOMETRY
AGGREGATE_VALUE		NUMBER

The theme_colname column must have a spatial index defined on it. For best performance, insert simplified geometries into this column.

Examples

The following example computes the thematic aggregates for all geometries in a table named TEST_TAB for an area with a 3-mile radius around a specified point geometry. In this case, the total population of each area is computed based on the proportion of the circle's area within different counties, assuming uniform distribution of population within the counties.

```
SELECT a.aggregate_value FROM TABLE(sdo_sam.aggregates_for_layer(
'GEOD_COUNTIES', 'GEOM', 'SUM', TOTPOP', TEST_TAB', 'GEOM'
'distance=3 unit=mile')) a;
```

SDO_SAM.BIN_GEOMETRY

Format

```
SDO_SAM.BIN_GEOMETRY(  
    geom          IN SDO_GEOMETRY,  
    tol          IN SDO_DIM_ARRAY,  
    bin_tablename IN VARCHAR2,  
    bin_colname  IN VARCHAR2  
    ) RETURN NUMBER;
```

or

```
SDO_SAM.BIN_GEOMETRY(  
    geom          IN SDO_GEOMETRY,  
    dim          IN SDO_DIM_ARRAY,  
    bin_tablename IN VARCHAR2,  
    bin_colname  IN VARCHAR2  
    ) RETURN NUMBER;
```

Description

Computes the most-intersecting tile for a geometry.

Parameters

geom

Geometry for which to compute the bin.

tol

Tolerance value (see [Section 1.5.5](#)).

dim

Dimensional array for the table that holds the geometries for the bins.

bin_tablename

Name of the table that holds the geometries for the bins.

bin_colname

Column in `bin_tablename` that holds the geometries for the bins.

Usage Notes

This function returns the bin that intersects most with the specified geometry. If multiple bins intersect to the same extent with the specified geometry, the bin with the smallest area is returned.

To perform this operation on all rows in the data mining table, using the specified `bin_tablename`, you can use the [SDO_SAM.BIN_LAYER](#) procedure.

Examples

The following example computes the bin for a specified geometry.

```
SELECT sdo_sam.bin_geometry(a.geometry, 0.0000005, 'BINTBL', 'GEOMETRY')
   FROM poly_4pt a, user_sdo_geom_metadata b
   WHERE b.table_name='POLY_4PT' AND a.gid=1;
```

```
SDO_SAM.BIN_GEOMETRY(A.GEOMETRY,0.0000005,'BINTBL','GEOMETRY')
```

```
-----  
43
```

```
1 row selected.
```

SDO_SAM.BIN_LAYER

Format

```
SDO_SAM.BIN_LAYER(  
    tablename      IN VARCHAR2,  
    colname        IN VARCHAR2,  
    bin_tablename  IN VARCHAR2,  
    bin_colname    IN VARCHAR2,  
    bin_id_colname IN VARCHAR2,  
    commit_interval IN NUMBER DEFAULT 20);
```

Description

Assigns each location (and the corresponding row) in a data mining table to a spatial bin.

Parameters

tablename

Name of the data mining table.

colname

Name of the column in `tablename` that holds the location coordinates.

bin_tablename

Name of the table that contains information (precomputed for the entire two-dimensional space) about the spatial bins.

bin_colname

Column in `bin_tablename` that holds the geometries for the bins.

bin_id_colname

Name of the column in the data mining table that holds the bin ID value of each geometry added to a bin. (Each affected row in the data mining table is updated with the ID value of the bin geometry in `bin_tablename`.)

commit_interval

Number of bin insert operations to perform before Spatial performs an internal commit operation. If `commit_interval` is not specified, a commit is performed after every 20 insert operations.

Usage Notes

This procedure computes the most-intersecting tile for each geometry in a specified layer using the bins in `bin_tablename`. The bin ID value for each geometry is added in `bin_id_colname`.

Using this procedure achieves the same result as using the [SDO_SAM.BIN_GEOMETRY](#) function on each row in the data mining table, using the specified `bin_tablename`.

Examples

The following example assigns each GEOMETRY column location and corresponding row in the POLY_4PT_TEMP data mining table to a spatial bin, and performs an internal commit operation after each bin table insertion.

```
CALL SDO_SAM.BIN_LAYER('POLY_4PT_TEMP', 'GEOMETRY', 'BINTBL', 'GEOMETRY', 'BIN_
ID', 1);
```

SDO_SAM.COLOCATED_REFERENCE_FEATURES

Format

```
SDO_SAM.COLOCATED_REFERENCE_FEATURES(  
    theme_tablename      IN VARCHAR2,  
    theme_colname        IN VARCHAR2,  
    theme_predicate      IN VARCHAR2,  
    tablename            IN VARCHAR2,  
    colname              IN VARCHAR2,  
    ref_predicate        IN VARCHAR2,  
    dst_spec             IN VARCHAR2,  
    result_tablename     IN VARCHAR2,  
    commit_interval     IN NUMBER DEFAULT 100);
```

Description

Performs a partial predicate-based join of tables, and materializes the join results into a table.

Parameters

theme_tablename

Name of the table with which to join `tablename`.

theme_colname

Name of the geometry column in `theme_tablename`.

theme_predicate

Qualifying WHERE clause predicate to be applied to `theme_tablename`.

tablename

Name of the data mining table.

colname

Name of the column in `tablename` that holds the location coordinates.

ref_predicate

Qualifying WHERE clause predicate to be applied to `tablename`. Must be a single table predicate, such as `'country_code=10'`.

dst_spec

A quoted string containing a distance value and optionally a unit value for a buffer around the geometries. See the Usage Notes for an explanation of the format and meaning.

result_tablename

The table in which materialized join results are stored. This table must have the following definition: `(tid NUMBER, rid1 VARCHAR2(24), rid2 VARCHAR2(24))`

commit_interval

Number of internal join operations to perform before Spatial performs an internal commit operation. If `commit_interval` is not specified, a commit is performed after every 100 internal join operations.

Usage Notes

This procedure materializes each pair of rowids returned from a predicate-based join operation, and stores them in the `rid1`, `rid2` columns of `result_tablename`. The `tid` is a unique generated "interaction" number corresponding to each `rid1` value.

The `dst_spec` parameter, if specified, is a quoted string containing the `distance` keyword and optionally the `unit` keyword (unit of measurement associated with the distance value), to specify a buffer around the geometry. For example, `'distance=2 unit=km'` specifies a 2-kilometer buffer around the input geometry. If `dst_spec` is not specified, no buffer is used.

If the `unit` keyword is specified, the value must be an `SDO_UNIT` value from the `MDSYS.SDO_DIST_UNITS` table (for example, `'unit=KM'`). If the `unit` keyword is not specified, the unit of measurement associated with the geometry is used. See [Section 2.8](#) for more information about unit of measurement specification.

Examples

The following example identifies cities with a 1990 population (`POP90` column value) greater than 120,000 that are located within 20 kilometers of interstate highways (`GEOM` column in the `GEOD_INTERSTATES` table). It stores the results in a table named `COLOCATION_TABLE`, and performs an internal commit operation after each 20 internal operations.

```
EXECUTE SDO_SAM.COLOCATED_REFERENCE_FEATURES(  
  'geod_cities', 'location', 'pop90 > 120000',  
  'geod_interstates', 'geom', null,  
  'distance=20 unit=km', 'colocation_table', 20);
```

SDO_SAM.SIMPLIFY_GEOMETRY

Format

```
SDO_SAM.SIMPLIFY_GEOMETRY(  
    geom            IN SDO_GEOMETRY,  
    dim            IN SDO_DIM_ARRAY,  
    pct_area_change_limit IN NUMBER DEFAULT 2  
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_SAM.SIMPLIFY_GEOMETRY(  
    geom            IN SDO_GEOMETRY,  
    tol            IN NUMBER,  
    pct_area_change_limit IN NUMBER DEFAULT 2  
    ) RETURN SDO_GEOMETRY;
```

Description

Simplifies a geometry.

Parameters

geom

Geometry to be simplified.

dim

Dimensional array for the geometry to be simplified.

tol

Tolerance value (see [Section 1.5.5](#)).

pct_area_change_limit

The percentage of area changed to be used for each simplification iteration, as explained in the Usage Notes.

Usage Notes

This function reduces the number of vertices in a geometry by internally applying the [SDO_UTIL.SIMPLIFY](#) function (documented in [Chapter 20](#)) with an appropriate threshold value.

Reducing the number of vertices may result in a change in the area of the geometry. The `pct_area_change_limit` parameter specifies how much area change can be tolerated while simplifying the geometry. It is usually a number from 1 to 100. The default value is 2; that is, the area of the geometry can either increase or decrease by at most two percent compared to the original geometry as a result of the geometry simplification.

Examples

The following example simplifies the geometries in the GEOMETRY column of the POLY_4PT_TEMP table.

```

SELECT sdo_sam.simplify_geometry(a.geometry, 0.00000005)
   FROM poly_4pt_temp a, user_sdo_geom_metadata b
   WHERE b.table_name='POLY_4PT_TEMP' ;

SDO_SAM.SIMPLIFY_GEOMETRY(A.GEOMETRY,0.00000005)(ORIG_AREA, CUR_AREA, ORIG_LEN,
-----
SDO_SMPL_GEOMETRY(28108.5905, 28108.5905, 758.440118, 758.440118, SDO_GEOMETRY(2
003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(-122.4215,
37.7862, -122.422, 37.7869, -122.421, 37.789, -122.42, 37.7866, -122.4215, 37.78
62)))

SDO_SMPL_GEOMETRY(4105.33806, 4105.33806, 394.723053, 394.723053, SDO_GEOMETRY(2
003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARRAY(-122.4019,
37.8052, -122.4027, 37.8055, -122.4031, 37.806, -122.4012, 37.8052, -122.4019, 3
7.8052)))
.
.
.
50 rows selected.

```

SDO_SAM.SIMPLIFY_LAYER

Format

```
SDO_SAM.SIMPLIFY_LAYER(  
    theme_tablename    IN VARCHAR2,  
    theme_colname      IN VARCHAR2,  
    smpl_geom_colname  IN VARCHAR2,  
    commit_interval    IN NUMBER DEFAULT 10,  
    pct_area_change_limit IN NUMBER DEFAULT 2);
```

Description

Simplifies a geometry layer.

Parameters

theme_tablename

Name of the table containing the geometry layer to be simplified.

theme_colname

Column in `theme_tablename` of type `SDO_GEOMETRY` containing the geometries to be simplified.

smpl_geom_colname

Column in `theme_tablename` of type `SDO_GEOMETRY` into which the simplified geometries are to be placed by this procedure.

commit_interval

Number of geometries to simplify before Spatial performs an internal commit operation. If `commit_interval` is not specified, a commit is performed after every 10 simplification operations.

pct_area_change_limit

The percentage of area changed to be used for each simplification iteration, as explained in the Usage Notes for the [SDO_SAM.SIMPLIFY_GEOMETRY](#) function.

Usage Notes

This procedure simplifies all geometries in a layer. It is equivalent to calling the [SDO_SAM.SIMPLIFY_GEOMETRY](#) function for each geometry in the layer, except that each simplified geometry is put in a separate column in the table instead of being returned to the caller. See also the Usage Notes for the [SDO_SAM.SIMPLIFY_GEOMETRY](#) function.

Examples

The following example adds a column named `SMPL_GEOM` to the `POLY_4PT_TEMP` table, then simplifies all geometries in the `GEOMETRY` column of the `POLY_4PT_TEMP` table, placing each simplified geometry in the `SMPL_GEOM` column in the same row with its associated original geometry.

```
ALTER TABLE poly_4pt_temp ADD (smpl_geom mdsys.sdo_geometry);
```

Table altered.

```
EXECUTE sdo_sam.simplify_layer('POLY_4PT_TEMP', 'GEOMETRY', 'SMPL_GEOM');
```

PL/SQL procedure successfully completed.

SDO_SAM.SPATIAL_CLUSTERS

Format

```
SDO_SAM.SPATIAL_CLUSTERS(  
    tablename    IN VARCHAR2,  
    colname      IN VARCHAR2,  
    max_clusters IN NUMBER,  
    allow_outliers IN VARCHAR2 DEFAULT 'TRUE',  
    tablepartition IN VARCHAR2 DEFAULT NULL  
) RETURN SDO_REGIONSET;
```

Description

Computes clusters using the existing R-tree index, and returns a set of SDO_REGION objects where the geometry column specifies the boundary of each cluster and the geometry_key value is set to null.

Parameters

tablename

Name of the data mining table.

colname

Name of the column in `tablename` that holds the location coordinates.

max_clusters

Maximum number of clusters to obtain.

allow_outliers

TRUE (the default) causes outlying values (isolated instances) to be included in the spatial clusters; FALSE causes outlying values not to be included in the spatial clusters. (TRUE accommodates all data and may result in larger clusters; FALSE may exclude some data and may result in smaller clusters.)

tablepartition

Name of the partition in `tablename`.

Usage Notes

The clusters are computed using the spatial R-tree index on `tablename`.

Examples

The following example clusters the locations in cities into at most three clusters, and includes outlying values in the clusters.

```
SELECT * FROM  
    TABLE(sdo_sam.spatial_clusters('PROJ_CITIES', 'LOCATION', 3, 'TRUE'));
```

SDO_SAM.TILED_AGGREGATES

Format

```
SDO_SAM.TILED_AGGREGATES(
    theme_name      IN VARCHAR2,
    theme_colname   IN VARCHAR2,
    aggr_type_string IN VARCHAR2,
    aggr_col_string IN VARCHAR2,
    tiling_level    IN NUMBER DEFAULT NULL,
    tiling_domain   IN SDO_DIM_ARRAY DEFAULT NULL,
    zero_agg_tiles  IN NUMBER DEFAULT 0,
    xdivs           IN NUMBER DEFAULT NULL,
    ydivs           IN NUMBER DEFAULT NULL
) RETURN SDO_REGAGGRSET;
```

Description

Tiles aggregates for a domain. For each tile, computes the intersecting geometries from the theme table; the values in the `aggr_col_string` column are weighted proportionally to the area of the intersection, and aggregated according to `aggr_col_string`.

Parameters

theme_name

Table containing theme information (for example, demographic information).

theme_colname

Name of the column in the `theme_name` table that contains geometry objects.

aggr_type_string

Any Oracle SQL aggregate function that accepts one or more numeric values and computes a numeric value, such as `SUM`, `MIN`, `MAX`, or `AVG`.

aggr_col_string

Name of a column in the `theme_name` table on which to compute aggregate values. An example might be a `POPULATION` column.

tiling_level

Level to be used to create tiles. If you specify this parameter, the extent of each dimension is divided into $2^{\text{tiling_level}}$ parts, resulting in at most $4^{\text{tiling_level}}$ tiles. (Specify either this parameter or the combination of the `xdivs` and `ydivs` parameters.)

tiling_domain

Domain for the tiling level. The parameter is not required, and if you do not specify it, the extent associated with the `theme_name` table is used.

zero_agg_tiles

Specify 0 to exclude tiles that have a value of 0 for the computed aggregate, or specify 1 to return all tiles. The default value is 0, which ensures that only tiles with a nonzero aggregate value are returned.

xdivs

The number of times that the extent in the first dimension is divided, such that the total number of parts is `xdivs + 1`. For example, if you specify 10 for `xdivs`, the extent of the first dimension is divided into 11 parts.

ydivs

The number of times that the extent in the second dimension is divided, such that the total number of parts is `ydivs + 1`. For example, if you specify 10 for `ydivs`, the extent of the second dimension is divided into 11 parts.

Usage Notes

This function is similar to [SDO_SAM.AGGREGATES_FOR_LAYER](#), but the results are dynamically generated using tiling information. Given a `theme_name` table, the tiling domain is determined. Based on the `tiling_level` value or the `xdivs` and `ydivs` values, the necessary tiles are generated. For each tile geometry, thematic aggregates are computed as described in the Usage Notes for [SDO_SAM.AGGREGATES_FOR_LAYER](#).

You must specify either the `tiling_level` parameter or both the `xdivs` and `ydivs` parameters. If you specify all three of these parameters, the `tiling_level` parameter is ignored and the `xdivs` and `ydivs` parameters are used.

If you specify the `xdivs` and `ydivs` parameters, the total number of grids (tiles) returned is $(xdivs+1) * (ydivs+1)$.

This function returns an object of type `SDO_REGAGGRSET`. The `SDO_REGAGGRSET` object type is defined as:

TABLE OF `SDO_REGAGGR`

The `SDO_REGAGGR` object type is defined as:

Name	Null?	Type
REGION_ID		VARCHAR2(24)
GEOMETRY		MDSYS.SDO_GEOMETRY
AGGREGATE_VALUE		NUMBER

Examples

The following example computes the sum of the population rows of `POLY_4PT_TEMP` table intersecting with each tile. The extent of the `POLY_4PT_TEMP` table stored in the `USER_SDO_GEOM_METADATA` view is used as the domain, and a tiling level of 2 is used (that is, the domain is divided into 16 tiles).

```
SELECT a.geometry, a.aggregate_value
  from TABLE(sdo_sam.tiled_aggregates('POLY_4PT_TEMP',
                                     'GEOMETRY', 'SUM', 'POPULATION', 2)) a;

GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
AGGREGATE_VALUE
-----
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(-180, -90, -90, -45))
```

```
.007150754

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARRAY(-180, -45, -90, 0))
.034831005

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARRAY(-180, 0, -90, 45))
7.73307783

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARRAY(-90, -90, 0, -45))
.019498368

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARRAY(-90, -45, 0, 0))
.939061456

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARRAY(-90, 0, 0, 45))
1.26691592

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARRAY(0, 0, 90, 45))
40
```

7 rows selected.

SDO_SAM.TILED_BINS

Format

```
SDO_SAM.TILED_BINS(  
    l1          IN NUMBER,  
    u1          IN NUMBER,  
    l2          IN NUMBER,  
    u2          IN NUMBER,  
    tiling_level IN NUMBER DEFAULT NULL,  
    srid        IN NUMBER DEFAULT NULL,  
    xdivs       IN NUMBER DEFAULT NULL,  
    ydivs       IN NUMBER DEFAULT NULL  
    ) RETURN SDO_REGIONSET;
```

Description

Tiles a two-dimensional space and returns geometries corresponding to those tiles.

Parameters

l1

Lower bound of the extent in the first dimension.

u1

Upper bound of the extent in the first dimension.

l2

Lower bound of the extent in the second dimension.

u2

Upper bound of the extent in the second dimension.

tiling_level

Level to be used to tile the specified extent. If you specify this parameter, the extent of each dimension is divided into $2^{\text{tiling_level}}$ parts, resulting in at most $4^{\text{tiling_level}}$ tiles. (Specify either this parameter or the combination of the `xdivs` and `ydivs` parameters.)

srid

SRID value to be included for the coordinate system in the returned tile geometries.

xdivs

The number of times that the extent in the first dimension is divided, such that the total number of parts is `xdivs + 1`. For example, if you specify 10 for `xdivs`, the extent of the first dimension is divided into 11 parts.

ydivs

The number of times that the extent in the second dimension is divided, such that the total number of parts is `ydivs + 1`. For example, if you specify 10 for `ydivs`, the extent of the second dimension is divided into 11 parts.

Usage Notes

You must specify either the `tiling_level` parameter or both the `xdivs` and `ydivs` parameters. If you specify all three of these parameters, the `tiling_level` parameter is ignored and the `xdivs` and `ydivs` parameters are used.

If you specify the `xdivs` and `ydivs` parameters, the total number of grids (tiles) returned is $(xdivs+1) * (ydivs+1)$.

This function returns an object of type `SDO_REGIONSET`. The `SDO_REGIONSET` object type is defined as:

TABLE OF `SDO_REGION`

The `SDO_REGION` object type is defined as:

Name	Null?	Type

ID		NUMBER
GEOMETRY		MDSYS.SDO_GEOMETRY

Examples

The following example tiles the entire Earth's surface at the first tiling level, using the standard longitude and latitude coordinate system (SRID 8307). The resulting `SDO_REGIONSET` object contains four `SDO_REGION` objects, one for each tile.

```
SELECT * FROM TABLE(sdo_sam.tiled_bins(-180, 180, -90, 90, 1, 8307))
ORDER BY id;

          ID
-----
GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
-----
          0
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(-180, -90, 0, 0))

          1
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(-180, 0, 0, 90))

          2
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(0, -90, 180, 0))

          3
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(0, 0, 180, 90))

4 rows selected.
```

SDO_TUNE Package (Tuning)

This chapter contains descriptions of the tuning subprograms shown in [Table 19-1](#).

Table 19-1 *Tuning Subprograms*

Subprogram	Description
SDO_TUNE.AVERAGE_MBR	Calculates the average minimum bounding rectangle for geometries in a layer.
SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE	Estimates the maximum number of megabytes needed for an R-tree spatial index table.
SDO_TUNE.EXTENT_OF	Returns the minimum bounding rectangle of the data in a layer.
SDO_TUNE.MIX_INFO	Calculates geometry type information for a spatial layer, such as the percentage of each geometry type.
SDO_TUNE.QUALITY_DEGRADATION	Returns the quality degradation for an index or the average quality degradation for all index tables for an index, or returns the quality degradation for a specified index table.

SDO_TUNE.AVERAGE_MBR

Format

```
SDO_TUNE.AVERAGE_MBR(  
    table_name  IN VARCHAR2,  
    column_name IN VARCHAR2,  
    width       OUT NUMBER,  
    height      OUT NUMBER);
```

Description

Calculates the average minimum bounding rectangle (MBR) for geometries in a layer.

Parameters

table_name

Spatial geometry table.

column_name

Geometry column for which the average minimum bounding rectangle is to be computed.

width

Width of the average minimum bounding rectangle.

height

Height of the average minimum bounding rectangle.

Usage Notes

This procedure computes and stores the width and height of the average minimum bounding rectangle for all geometries in a spatial geometry table. It calculates the average MBR by keeping track of the maximum and minimum X and Y values for all geometries in a spatial geometry table.

Examples

The following example calculates the minimum bounding rectangle for the SHAPE column of the COLA_MARKETS table.

```
DECLARE  
    table_name VARCHAR2(32) := 'COLA_MARKETS';  
    column_name VARCHAR2(32) := 'SHAPE';  
    width NUMBER;  
    height NUMBER;  
BEGIN  
    SDO_TUNE.AVERAGE_MBR(  
        table_name,  
        column_name,  
        width,  
        height);  
    DBMS_OUTPUT.PUT_LINE('Width = ' || width);  
    DBMS_OUTPUT.PUT_LINE('Height = ' || height);  
END;
```


/
Width = 3.5
Height = 4.5

Related Topics

[SDO_AGGR_MBR](#) spatial aggregate function (in [Chapter 12](#))

SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE

Format

```
SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE(  
    schemaname IN VARCHAR2,  
    tabname    IN VARCHAR2,  
    colname    IN VARCHAR2,  
    partname   IN VARCHAR2 DEFAULT NULL  
    ) RETURN NUMBER;
```

or

```
SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE(  
    number_of_geoms IN INTEGER,  
    db_block_size   IN INTEGER,  
    sdo_rtr_pctfree IN INTEGER DEFAULT 10,  
    num_dimensions  IN INTEGER DEFAULT 2,  
    is_geodetic     IN INTEGER DEFAULT 0  
    ) RETURN NUMBER;
```

Description

Estimates the maximum number of megabytes needed for an R-tree spatial index table.

Parameters

schemaname

Schema that owns the spatial geometry table.

tabname

Spatial geometry table name.

colname

Geometry column name.

partname

Name of a partition containing geometries from `colname`. If you specify this parameter, the value returned by the function is the estimated size for an R-tree index table on geometries in that partition. If you do not specify this parameter, the value is the estimated size for an R-tree index table on all geometries in `colname`.

number_of_geoms

Approximate number of geometries in the spatial geometry table.

db_block_size

Database block size (in bytes).

sdo_rtr_pctfree

Minimum percentage of slots in each index tree node to be left empty when the index is created. Slots that are left empty can be filled later when new data is inserted into

the table. The value can range from 0 to 50. The default value (10) is best for most applications; however, a value of 0 is recommended if no updates will be performed to the geometry column.

num_dimensions

Number of dimensions to be indexed. The default value is 2. If you plan to specify the `sdo_indx_dims` parameter in the [CREATE INDEX](#) statement, the `num_dimensions` value should match the `sdo_indx_dims` value.

is_geodetic

A value indicating whether or not the spatial index will be a geodetic index: 1 for a geodetic index, or 0 (the default) for a non-geodetic index. ([Section 4.1.1](#) explains geodetic indexes.)

Usage Notes

The function returns the estimated maximum number of megabytes needed for the spatial index table (described in [Section 2.7.2](#)) for an R-tree spatial index to be created. The value returned is the maximum number of megabytes needed after index creation. During index creation, approximately three times this value of megabytes will be needed in the tablespace, to ensure that there is enough space for temporary tables while the index is being created.

This function has two formats:

- Use the format with character string parameters (`schemaname`, `tablename`, `colname`, and optionally `partname`) in most cases when the spatial geometry table already exists, you do not plan to add substantially more geometries to it before creating the index, and you plan to use the default R-tree indexing parameters.
- Use the format with integer parameters (`number_of_geoms`, `db_block_size`, `sdo_rtr_pctfree`, `num_dimensions`, `is_geodetic`) in any of the following cases: the spatial geometry table does not exist; the spatial geometry table exists but you plan to add substantially more geometries to it before creating the index; the `num_dimensions` value is not 2 for non-geodetic data or 3 for geodetic data, and a nondefault value will be specified using the `sdo_indx_dims` parameter in the [CREATE INDEX](#) statement; or the data is geodetic but you plan to specify `'geodetic=false'` in the [CREATE INDEX](#) statement (see [Section 4.1.1](#)).

Examples

The following example estimates the maximum number of megabytes needed for a spatial index table for an index given the following information: `number_of_geoms = 1000000` (one million), `db_block_size = 2048`, `sdo_rtr_pctfree = 10`, `num_dimensions = 2`, `is_geodetic = 0`.

```
SELECT SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE(1000000, 2048, 10, 2, 0) FROM DUAL;
```

```
SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE(1000000,2048,10,2,0)
```

```
-----  
82
```

The following example estimates the maximum number of megabytes needed for a spatial index table for an index on the `SHAPE` column in the `COLA_MARKETS` table in the `SCOTT` schema. The estimate is based on the geometries that are currently in the table.

```
SELECT SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE('SCOTT', 'COLA_MARKETS', 'SHAPE') FROM DUAL;
```

```
SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE('SCOTT', 'COLA_MARKETS', 'SHAPE')
```

1

SDO_TUNE.EXTENT_OF

Format

```
SDO_TUNE.EXTENT_OF(
    table_name  IN VARCHAR2,
    column_name IN VARCHAR2
) RETURN SDO_GEOMETRY;
```

Description

Returns the minimum bounding rectangle (MBR) of all geometries in a layer.

Parameters

table_name

Spatial geometry table.

column_name

Geometry column for which the minimum bounding rectangle is to be returned.

Usage Notes

The [SDO_AGGR_MBR](#) function, documented in [Chapter 12](#), also returns the MBR of geometries. The SDO_TUNE.EXTENT_OF function has better performance than the [SDO_AGGR_MBR](#) function if the data is non-geodetic and if a spatial index is defined on the geometry column; however, the SDO_TUNE.EXTENT_OF function is limited to two-dimensional geometries, whereas the [SDO_AGGR_MBR](#) function is not. In addition, the SDO_TUNE.EXTENT_OF function computes the extent for all geometries in a table; by contrast, the [SDO_AGGR_MBR](#) function can operate on subsets of rows.

The SDO_TUNE.EXTENT_OF function returns NULL if the data is inconsistent.

If a non-geodetic spatial index is used, this function may return an approximate MBR that encloses the largest extent of data stored in the index, even if data was subsequently deleted. This can occur because the function extracts MBR information from a non-geodetic spatial index, if one exists.

Examples

The following example calculates the minimum bounding rectangle for the objects in the SHAPE column of the COLA_MARKETS table.

```
SELECT SDO_TUNE.EXTENT_OF('COLA_MARKETS', 'SHAPE')
       FROM DUAL;
```

```
SDO_TUNE.EXTENT_OF('COLA_MARKETS', 'SHAPE')(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y,
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_
ARRAY(1, 1, 10, 11))
```

Related Topics

[SDO_AGGR_MBR](#) aggregate function (in [Chapter 12](#))

[SDO_TUNE.AVERAGE_MBR](#) procedure

SDO_TUNE.MIX_INFO

Format

```
SDO_TUNE.MIX_INFO(  
    table_name    IN VARCHAR2,  
    column_name   IN VARCHAR2  
    [, total_geom OUT INTEGER,  
    point_geom    OUT INTEGER,  
    curve_geom    OUT INTEGER,  
    poly_geom     OUT INTEGER,  
    complex_geom  OUT INTEGER] );
```

Description

Calculates geometry type information for a spatial layer, such as the percentage of each geometry type.

Parameters

table_name

Spatial geometry table.

column_name

Geometry object column for which the geometry type information is to be calculated.

total_geom

Total number of geometry objects.

point_geom

Number of point geometry objects.

curve_geom

Number of curve string geometry objects.

poly_geom

Number of polygon geometry objects.

complex_geom

Number of complex geometry objects.

Usage Notes

This procedure calculates geometry type information for the table. It calculates the total number of geometries, as well as the number of point, curve string, polygon, and complex geometries.

Examples

The following example displays information about the mix of geometry objects in the SHAPE column of the COLA_MARKETS table.

```
CALL SDO_TUNE.MIX_INFO('COLA_MARKETS', 'SHAPE');
```

Total number of geometries: 4
Point geometries: 0 (0%)
Curvestring geometries: 0 (0%)
Polygon geometries: 4 (100%)
Complex geometries: 0 (0%)

SDO_TUNE.QUALITY_DEGRADATION

Format

```
SDO_TUNE.QUALITY_DEGRADATION(  
    schemaname IN VARCHAR2,  
    indexname   IN VARCHAR2  
    ) RETURN NUMBER;
```

or

```
SDO_TUNE.QUALITY_DEGRADATION(  
    schemaname IN VARCHAR2,  
    indexname   IN VARCHAR2,  
    indextable  IN VARCHAR2  
    ) RETURN NUMBER;
```

Description

Returns the quality degradation for an index or the average quality degradation for all index tables for an index, or returns the quality degradation for a specified index table.

Parameters

schemaname

Name of the schema that contains the index specified in `indexname`.

indexname

Name of the spatial R-tree index.

indextable

Name of an index table associated with the spatial R-tree index specified in `indexname`.

Usage Notes

The **quality degradation** is a number indicating approximately how much longer it will take to execute the I/O operations of the index portion of any given query with the current index, compared to executing the I/O operations of the index portion of the same query when the index was created or most recently rebuilt. For example, if the I/O operations of the index portion of a typical query will probably take twice as much time as when the index was created or rebuilt, the quality degradation is 2. The exact degradation in overall query time is impossible to predict; however, a substantial quality degradation (2 or 3 or higher) can affect query performance significantly for large databases, such as those with millions of rows.

For local partitioned indexes, you must use the format that includes the `indextable` parameter. That is, you must compute the quality degradation for each partition in which you are interested.

Index names are available through the `xxx_SDO_INDEX_INFO` and `xxx_SDO_INDEX_METADATA` views, which are described in [Section 2.7.1](#).

For more information and guidelines relating to R-tree quality and its possible effect on query performance, see [Section 1.7.2](#).

Examples

The following example returns the quality degradation for the COLA_SPATIAL_IDX index. In this example, the quality has not degraded at all, and therefore the degradation is 1; that is, the I/O operations of the index portion of queries will typically take the same time using the current index as using the original or previous index.

```
SELECT SDO_TUNE.QUALITY_DEGRADATION('SCOTT', 'COLA_SPATIAL_IDX') FROM DUAL;
```

```
SDO_TUNE.QUALITY_DEGRADATION('SCOTT', 'COLA_SPATIAL_IDX')
```

```
-----
```

```
1
```

SDO_UTIL Package (Utility)

This chapter contains descriptions of the spatial utility subprograms shown in Table 20-1.

Table 20-1 Spatial Utility Subprograms

Subprogram	Description
SDO_UTIL.APPEND	Appends one geometry to another geometry to create a new geometry.
SDO_UTIL.CIRCLE_POLYGON	Returns the polygon geometry that approximates and is covered by a specified circle.
SDO_UTIL.CONCAT_LINES	Concatenates two line or multiline two-dimensional geometries to create a new geometry.
SDO_UTIL.CONVERT_UNIT	Converts values from one angle, area, or distance unit of measure to another.
SDO_UTIL.ELLIPSE_POLYGON	Returns the polygon geometry that approximates and is covered by a specified ellipse.
SDO_UTIL.EXTRACT	Returns the geometry that represents a specified element (and optionally a ring) of the input geometry.
SDO_UTIL.FROM_WKBGEOMETRY	Converts a geometry in the well-known binary (WKB) format to a Spatial geometry object.
SDO_UTIL.FROM_WKTGEOMETRY	Converts a geometry in the well-known text (WKT) format to a Spatial geometry object.
SDO_UTIL.GETNUMELEM	Returns the number of elements in the input geometry.
SDO_UTIL.GETNUMVERTICES	Returns the number of vertices in the input geometry.
SDO_UTIL.GETVERTICES	Returns the coordinates of the vertices of the input geometry.
SDO_UTIL.INITIALIZE_INDEXES_FOR_TTS	Initializes all spatial indexes in a tablespace that was transported to another database.
SDO_UTIL.POINT_AT_BEARING	Returns a point geometry that is at the specified distance and bearing from the start point.
SDO_UTIL.POLYGONTOLINE	Converts all polygon-type elements in a geometry to line-type elements, and sets the SDO_GTYPE value accordingly.
SDO_UTIL.PREPARE_FOR_TTS	Prepares a tablespace to be transported to another database, so that spatial indexes will be preserved during the transport operation.
SDO_UTIL.RECTIFY_GEOMETRY	Fixes certain problems with the input geometry, and returns a valid geometry.

Table 20–1 (Cont.) Spatial Utility Subprograms

Subprogram	Description
<code>SDO_UTIL.REMOVE_DUPLICATE_VERTICES</code>	Removes duplicate (redundant) vertices from a geometry.
<code>SDO_UTIL.REVERSE_LINESTRING</code>	Returns a line string geometry with the vertices of the input geometry in reverse order.
<code>SDO_UTIL.SIMPLIFY</code>	Simplifies the input geometry, based on a threshold value, using the Douglas-Peucker algorithm.
<code>SDO_UTIL.TO_GMLGEOMETRY</code>	Converts a Spatial geometry object to a geography markup language (GML 2.0) fragment based on the geometry types defined in the Open GIS <code>geometry.xsd</code> schema document.
<code>SDO_UTIL.TO_WKBGEOMETRY</code>	Converts a Spatial geometry object to the well-known binary (WKB) format.
<code>SDO_UTIL.TO_WKTGEOMETRY</code>	Converts a Spatial geometry object to the well-known text (WKT) format.
<code>SDO_UTIL.VALIDATE_WKBGEOMETRY</code>	Validates the input geometry, which is in the standard well-known binary (WKB) format; returns the string <code>TRUE</code> if the geometry is valid or <code>FALSE</code> if the geometry is not valid.
<code>SDO_UTIL.VALIDATE_WKTGEOMETRY</code>	Validates the input geometry, which is of type <code>CLOB</code> or <code>VARCHAR2</code> and in the standard well-known text (WKT) format; returns the string <code>TRUE</code> if the geometry is valid or <code>FALSE</code> if the geometry is not valid.

SDO_UTIL.APPEND

Format

```
SDO_UTIL.APPEND(
    geom1 IN SDO_GEOMETRY,
    geom2 IN SDO_GEOMETRY
) RETURN SDO_GEOMETRY;
```

Description

Appends one geometry to another geometry to create a new geometry.

Parameters

geom1

Geometry object to which geom2 is to be appended.

geom2

Geometry object to append to geom1.

Usage Notes

This function should be used only on geometries that do not have any spatial interaction (that is, on disjoint objects). If the input geometries are not disjoint, the resulting geometry might be invalid.

This function does not perform a union operation or any other computational geometry operation. To perform a union operation, use the [SDO_GEOM.SDO_UNION](#) function, which is described in [Chapter 15](#). The APPEND function executes faster than the [SDO_GEOM.SDO_UNION](#) function.

The geometry type (SDO_GTYPE value) of the resulting geometry reflects the types of the input geometries and the append operation. For example, if the input geometries are two-dimensional polygons (SDO_GTYPE = 2003), the resulting geometry is a two-dimensional multipolygon (SDO_GTYPE = 2007).

An exception is raised if geom1 and geom2 are based on different coordinate systems.

Examples

The following example appends the cola_a and cola_c geometries. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT SDO_UTIL.APPEND(c_a.shape, c_c.shape)
   FROM cola_markets c_a, cola_markets c_c
   WHERE c_a.name = 'cola_a' AND c_c.name = 'cola_c';

SDO_UTIL.APPEND(C_A.SHAPE,C_C.SHAPE)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SD
-----
SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3, 5, 1003, 1), SDO_
ORDINATE_ARRAY(1, 1, 5, 7, 3, 3, 6, 3, 6, 5, 4, 5, 3, 3))
```

Related Topics

- [SDO_GEOM.SDO_UNION](#) (in [Chapter 15](#))

SDO_UTIL.CIRCLE_POLYGON

Format

```
SDO_UTIL.CIRCLE_POLYGON(
  center_longitude IN NUMBER,
  center_latitude  IN NUMBER,
  radius           IN NUMBER,
  arc_tolerance    IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns the polygon geometry that approximates and is covered by a specified circle.

Parameters

center_longitude

Center longitude (in degrees) of the circle to be used to create the returned geometry.

center_latitude

Center latitude (in degrees) of the circle to be used to create the returned geometry.

radius

Length (in meters) of the radius of the circle to be used to create the returned geometry.

arc_tolerance

A numeric value to be used to construct the polygon geometry. The `arc_tolerance` parameter value has the same meaning and usage guidelines as the `arc_tolerance` keyword value in the `params` parameter string for the [SDO_GEOM.SDO_ARC_DENSIFY](#) function. The unit of measurement associated with the geometry is associated with the `arc_tolerance` parameter value. (For more information, see the Usage Notes for the [SDO_GEOM.SDO_ARC_DENSIFY](#) function in [Chapter 15](#).)

Usage Notes

This function is useful for creating a circle-like polygon around a specified center point when a true circle cannot be used (a circle is not valid for geodetic data with Oracle Spatial). The returned geometry has an `SDO_SRID` value of 8307 (for Longitude / Latitude (WGS 84)).

Examples

The following example returns a circle-like polygon around a point near the center of Concord, Massachusetts. A `radius` value of 100 meters and an `arc_tolerance` value of 5 meters are used in computing the polygon vertices.

```
SELECT SDO_UTIL.CIRCLE_POLYGON(-71.34937, 42.46101, 100, 5)
       FROM DUAL;

SDO_UTIL.CIRCLE_POLYGON(-71.34937,42.46101,100,5) (SDO_GTYPE, SDO_SRID, SDO_POINT
-----
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
```

```
AY(-71.34937, 42.4601107, -71.348653, 42.4602824, -71.348211, 42.4607321, -71.34  
8211, 42.4612879, -71.348653, 42.4617376, -71.34937, 42.4619093, -71.350087, 42.  
4617376, -71.350529, 42.4612879, -71.350529, 42.4607321, -71.350087, 42.4602824,  
-71.34937, 42.4601107))
```

Related Topics

- [SDO_UTIL.ELLIPSE_POLYGON](#)

SDO_UTIL.CONCAT_LINES

Format

```
SDO_UTIL.CONCAT_LINES(
  geom1 IN SDO_GEOMETRY,
  geom2 IN SDO_GEOMETRY
) RETURN SDO_GEOMETRY;
```

Description

Concatenates two line or multiline two-dimensional geometries to create a new geometry.

Parameters

geom1

First geometry object for the concatenation operation.

geom2

Second geometry object for the concatenation operation.

Usage Notes

Each input geometry must be a two-dimensional line or multiline geometry (that is, the `SDO_GTYPE` value must be 2002 or 2006). This function is not supported for LRS geometries. To concatenate LRS geometric segments, use the [SDO_LRS.CONCATENATE_GEOM_SEGMENTS](#) function (described in [Chapter 16](#)).

The input geometries must be line strings whose vertices are connected by straight line segments. Circular arcs and compound line strings are not supported.

If an input geometry is a multiline geometry, the elements of the geometry must be disjoint. If they are not disjoint, this function may return incorrect results.

The topological relationship between `geom1` and `geom2` must be `DISJOINT` or `TOUCH`; and if the relationship is `TOUCH`, the geometries must intersect only at two end points.

You can use the [SDO_AGGR_CONCAT_LINES](#) spatial aggregate function (described in [Chapter 12](#)) to concatenate multiple two-dimensional line or multiline geometries.

An exception is raised if `geom1` and `geom2` are based on different coordinate systems.

Examples

The following example concatenates two simple line string geometries.

```
-- Concatenate two touching lines: one from (1,1) to (5,1) and the
-- other from (5,1) to (8,1).
SELECT SDO_UTIL.CONCAT_LINES(
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,2,1),
    SDO_ORDINATE_ARRAY(1,1, 5,1)),
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,2,1),
    SDO_ORDINATE_ARRAY(5,1, 8,1))
) FROM DUAL;

SDO_UTIL.CONCAT_LINES(SDO_GEOMETRY(2002,NULL,NULL,SDO_ELEM_INFO_ARRAY(1,2,1),SDO
```



```
-----  
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(  
1, 1, 5, 1, 8, 1))
```

Related Topics

- [SDO_AGGR_CONCAT_LINES](#) (in Chapter 12)
- [SDO_LRS.CONCATENATE_GEOM_SEGMENTS](#) (in Chapter 16)

SDO_UTIL.CONVERT_UNIT

Format

```
SDO_UTIL.CONVERT_UNIT(  
    input_value IN NUMBER,  
    from_unit   IN VARCHAR2,  
    to_unit     IN VARCHAR2  
) RETURN NUMBER;
```

Description

Converts values from one angle, area, or distance unit of measure to another.

Parameters

input_value

Number of units to be converted. For example, to convert 10 decimal degrees to radians, specify 10.

from_unit

The unit of measure from which to convert the input value. Must be a value from the SDO_UNIT column of the MDSYS.SDO_ANGLE_UNITS table (described in [Section 6.7.2](#)), the MDSYS.SDO_AREA_UNITS table (described in [Section 2.8](#)), or the MDSYS.SDO_DIST_UNITS table (described in [Section 2.8](#)). For example, to convert decimal degrees to radians, specify Degree.

to_unit

The unit of measure into which to convert the input value. Must be a value from the SDO_UNIT column of the same table used for from_unit. For example, to convert decimal degrees to radians, specify Radian.

Usage Notes

The value returned by this function might not be correct at an extremely high degree of precision because of the way internal mathematical operations are performed, especially if they involve small numbers or irrational numbers (such as π). For example, converting 1 decimal degree into decimal minutes results in the value 60.0000017.

Examples

The following example converts 1 radian into decimal degrees.

```
SQL> SELECT SDO_UTIL.CONVERT_UNIT(1, 'Radian', 'Degree') FROM DUAL;
```

```
SDO_UTIL.CONVERT_UNIT(1, 'RADIAN', 'DEGREE')  
-----  
57.2957796
```

Related Topics

None.

SDO_UTIL.ELLIPSE_POLYGON

Format

```
SDO_UTIL.ELLIPSE_POLYGON(  
    center_longitude IN NUMBER,  
    center_latitude  IN NUMBER,  
    semi_major_axis  IN NUMBER,  
    semi_minor_axis  IN NUMBER,  
    azimuth          IN NUMBER,  
    arc_tolerance    IN NUMBER  
) RETURN SDO_GEOMETRY;
```

Description

Returns the polygon geometry that approximates and is covered by a specified ellipse.

Parameters

center_longitude

Center longitude (in degrees) of the ellipse to be used to create the returned geometry.

center_latitude

Center latitude (in degrees) of the ellipse to be used to create the returned geometry.

semi_major_axis

Length (in meters) of the semi-major axis of the ellipse to be used to create the returned geometry.

semi_minor_axis

Length (in meters) of the semi-minor axis of the ellipse to be used to create the returned geometry.

azimuth

Number of degrees of the azimuth (clockwise rotation of the major axis from north) of the ellipse to be used to create the returned geometry. Must be from 0 to 180. The returned geometry is rotated by the specified number of degrees.

arc_tolerance

A numeric value to be used to construct the polygon geometry. The `arc_tolerance` parameter value has the same meaning and usage guidelines as the `arc_tolerance` keyword value in the `params` parameter string for the [SDO_GEOM.SDO_ARC_DENSIFY](#) function. The unit of measurement associated with the geometry is associated with the `arc_tolerance` parameter value. (For more information, see the Usage Notes for the [SDO_GEOM.SDO_ARC_DENSIFY](#) function in [Chapter 15](#).)

Usage Notes

This function is useful for creating an ellipse-like polygon around a specified center point when a true ellipse cannot be used (an ellipse is not valid for geodetic data with Oracle Spatial). The returned geometry has an `SDO_SRID` value of 8307 (for Longitude / Latitude (WGS 84)).

Examples

The following example returns an ellipse-like polygon, oriented east-west (azimuth = 90), around a point near the center of Concord, Massachusetts. An `arc_tolerance` value of 5 meters is used in computing the polygon vertices.

```
SELECT SDO_UTIL.ELLIPSE_POLYGON(-71.34937, 42.46101, 100, 50, 90, 5)
       FROM DUAL;

SDO_UTIL.ELLIPSE_POLYGON(-71.34937,42.46101,100,50,90,5) (SDO_GTYPE, SDO_SRID, SD
-----
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(-71.350589, 42.46101, -71.350168, 42.4606701, -71.349708, 42.460578, -71.3493
7, 42.4605603, -71.349032, 42.460578, -71.348572, 42.4606701, -71.348151, 42.461
01, -71.348572, 42.4613499, -71.349032, 42.461442, -71.34937, 42.4614597, -71.34
9708, 42.461442, -71.350168, 42.4613499, -71.350589, 42.46101))
```

Related Topics

- [SDO_UTIL.CIRCLE_POLYGON](#)

SDO_UTIL.EXTRACT

Format

```
SDO_UTIL.EXTRACT(
    geometry IN SDO_GEOMETRY,
    element  IN NUMBER
    [, ring   IN NUMBER]
) RETURN SDO_GEOMETRY;
```

Description

Returns the geometry that represents a specified element (and optionally a ring) of the input geometry.

Parameters

geometry

Geometry from which to extract the geometry to be returned.

element

Number of the element in the geometry: 1 for the first element, 2 for the second element, and so on. Geometries with SDO_GTYPE values (explained in [Section 2.2.1](#)) ending in 1, 2, or 3 have one element; geometries with SDO_GTYPE values ending in 4, 5, 6, or 7 can have more than one element. For example, a multipolygon with an SDO_GTYPE of 2007 might contain three elements (polygons).

ring

Number of the subelement (ring) within `element`: 1 for the first subelement, 2 for the second subelement, and so on. This parameter is valid only for specifying a subelement of a polygon with one or more holes or of a point cluster:

- For a polygon with holes, its first subelement is its exterior ring, its second subelement is its first interior ring, its third subelement is its second interior ring, and so on. For example, in the polygon with a hole shown in [Figure 2-3](#) in [Section 2.5.2](#), the exterior ring is subelement 1 and the interior ring (the hole) is subelement 2.
- For a point cluster, its first subelement is the first point in the point cluster, its second subelement is the second point in the point cluster, and so on.

The default is 0, which causes the entire element to be extracted.

Usage Notes

This function is useful for extracting a specific element or subelement from a complex geometry. For example, if you have identified a geometry as invalid by using the [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#) function or the [SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT](#) procedure (both of which are documented in [Chapter 15](#)), you can use the EXTRACT function to extract the invalid geometry in order to examine it.

For a polygon with one or more holes, the returned geometry representing an extracted interior ring is reoriented so that its vertices are presented in counterclockwise order (as opposed to the clockwise order within an interior ring).

If `geometry` is null or has an `SDO_GTYPE` value ending in 0, this function returns a null geometry.

`geometry` cannot contain a type 0 (zero) element. Type 0 elements are described in [Section 2.5.7](#).

An exception is raised if `element` or `ring` is an invalid number for geometry.

Examples

The following example extracts the first (and only) element in the `cola_c` geometry. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT c.name, SDO_UTIL.EXTRACT(c.shape, 1)
       FROM cola_markets c WHERE c.name = 'cola_c';
```

NAME

```
-----
SDO_UTIL.EXTRACT(C.SHAPE,1)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_IN
-----
cola_c
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(3, 3, 6, 3, 6, 5, 4, 5, 3, 3))
```

The following example inserts a polygon with a hole (using the same INSERT statement as in [Example 2–5](#) in [Section 2.5.2](#)), and extracts the geometry representing the hole (the second subelement). Notice that in the geometry returned by the EXTRACT function, the vertices are in counterclockwise order, as opposed to the clockwise order in the hole (second subelement) in the input geometry.

```
-- Insert polygon with hole.
INSERT INTO cola_markets VALUES(
  10,
  'polygon_with_hole',
  SDO_GEOMETRY(
    2003, -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1, 19,2003,1), -- polygon with hole
    SDO_ORDINATE_ARRAY(2,4, 4,3, 10,3, 13,5, 13,9, 11,13, 5,13, 2,11, 2,4,
      7,5, 7,10, 10,10, 10,5, 7,5)
  )
);

1 row created.

-- Extract the hole geometry (second subelement).
SELECT SDO_UTIL.EXTRACT(c.shape, 1, 2)
       FROM cola_markets c WHERE c.name = 'polygon_with_hole';

SDO_UTIL.EXTRACT(C.SHAPE,1,2)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(7, 5, 10, 5, 10, 10, 7, 10, 7, 5))
```

Related Topics

- [SDO_UTIL.GETVERTICES](#)
- [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#)
- [SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT](#)

SDO_UTIL.FROM_WKBGEOMETRY

Format

```
SDO_UTIL.FROM_WKBGEOMETRY(
    geometry IN BLOB
) RETURN SDO_GEOMETRY;
```

Description

Converts a geometry in the well-known binary (WKB) format to a Spatial geometry object.

Parameters

geometry

Geometry in WKB format to be converted to SDO_GEOMETRY format.

Usage Notes

The input geometry must be in the well-known binary (WKB) format, as defined by the Open Geospatial Consortium and the International Organization for Standardization (ISO).

This function is patterned after the SQL Multimedia recommendations in *ISO 13249-3, Information technology - Database languages - SQL Multimedia and Application Packages - Part 3: Spatial*.

To convert an SDO_GEOMETRY object to WKB format, use the [SDO_UTIL.TO_WKBGEOMETRY](#) function.

Examples

The following example shows conversion to and from WKB and WKT format, and validation of WKB and WKT geometries. (The example uses the definitions and data from [Section 2.1](#), specifically the `cola_b` geometry from the `COLA_MARKETS` table.)

```
DECLARE
    wkbgeom BLOB;
    wktgeom CLOB;
    val_result VARCHAR2(5);
    geom_result SDO_GEOMETRY;
    geom SDO_GEOMETRY;
BEGIN
    SELECT c.shape INTO geom FROM cola_markets c WHERE c.name = 'cola_b';

    -- To WBT/WKT geometry
    wkbgeom := SDO_UTIL.TO_WKBGEOMETRY(geom);
    wktgeom := SDO_UTIL.TO_WKTGEOMETRY(geom);
    DBMS_OUTPUT.PUT_LINE('To WKT geometry result = ' || TO_CHAR(wktgeom));

    -- From WBT/WKT geometry
    geom_result := SDO_UTIL.FROM_WKBGEOMETRY(wkbgeom);
    geom_result := SDO_UTIL.FROM_WKTGEOMETRY(wktgeom);

    -- Validate WBT/WKT geometry
    val_result := SDO_UTIL.VALIDATE_WKBGEOMETRY(wkbgeom);
```

```
DBMS_OUTPUT.PUT_LINE('WKB validation result = ' || val_result);
val_result := SDO_UTIL.VALIDATE_WKTGEOMETRY(wktgeom);
DBMS_OUTPUT.PUT_LINE('WKT validation result = ' || val_result);

END;
/
```

```
To WKT geometry result = POLYGON ((5.0 1.0, 8.0 1.0, 8.0 6.0, 5.0 7.0, 5.0 1.0))
WKB validation result = TRUE
WKT validation result = TRUE
```

Related Topics

- [SDO_UTIL.FROM_WKTGEOMETRY](#)
- [SDO_UTIL.TO_WKBGEOMETRY](#)
- [SDO_UTIL.TO_WKTGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKBGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKTGEOMETRY](#)

SDO_UTIL.FROM_WKTGEOMETRY

Format

```
SDO_UTIL.FROM_WKTGEOMETRY(
    geometry IN CLOB
) RETURN SDO_GEOMETRY;
```

or

```
SDO_UTIL.FROM_WKTGEOMETRY(
    geometry IN VARCHAR2
) RETURN SDO_GEOMETRY;
```

Description

Converts a geometry in the well-known text (WKT) format to a Spatial geometry object.

Parameters

geometry

Geometry in WKT format to be converted to SDO_GEOMETRY format.

Usage Notes

The input geometry must be in the well-known text (WKT) format, as defined by the Open Geospatial Consortium and the International Organization for Standardization (ISO).

This function is patterned after the SQL Multimedia recommendations in *ISO 13249-3, Information technology - Database languages - SQL Multimedia and Application Packages - Part 3: Spatial*.

To convert an SDO_GEOMETRY object to a CLOB in WKT format, use the [SDO_UTIL.TO_WKTGEOMETRY](#) function. (You can use the SQL function TO_CHAR to convert the resulting CLOB to VARCHAR2 type.)

Examples

The following example shows conversion to and from WKB and WKT format, and validation of WKB and WKT geometries. (The example uses the definitions and data from [Section 2.1](#), specifically the cola_b geometry from the COLA_MARKETS table.)

```
DECLARE
    wkbgeom BLOB;
    wktgeom CLOB;
    val_result VARCHAR2(5);
    geom_result SDO_GEOMETRY;
    geom SDO_GEOMETRY;
BEGIN
    SELECT c.shape INTO geom FROM cola_markets c WHERE c.name = 'cola_b';

    -- To WBT/WKT geometry
    wkbgeom := SDO_UTIL.TO_WKBGEOMETRY(geom);
    wktgeom := SDO_UTIL.TO_WKTGEOMETRY(geom);
    DBMS_OUTPUT.PUT_LINE('To WKT geometry result = ' || TO_CHAR(wktgeom));
```

```
-- From WBT/WKT geometry
geom_result := SDO_UTIL.FROM_WKBGEOMETRY(wkbgeom);
geom_result := SDO_UTIL.FROM_WKTGEOMETRY(wktgeom);

-- Validate WBT/WKT geometry
val_result := SDO_UTIL.VALIDATE_WKBGEOMETRY(wkbgeom);
DBMS_OUTPUT.PUT_LINE('WKB validation result = ' || val_result);
val_result := SDO_UTIL.VALIDATE_WKTGEOMETRY(wktgeom);
DBMS_OUTPUT.PUT_LINE('WKT validation result = ' || val_result);

END;
/

To WKT geometry result = POLYGON ((5.0 1.0, 8.0 1.0, 8.0 6.0, 5.0 7.0, 5.0 1.0))
WKB validation result = TRUE
WKT validation result = TRUE
```

Related Topics

- [SDO_UTIL.FROM_WKBGEOMETRY](#)
- [SDO_UTIL.TO_WKBGEOMETRY](#)
- [SDO_UTIL.TO_WKTGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKBGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKTGEOMETRY](#)

SDO_UTIL.GETNUMELEM

Format

```
SDO_UTIL.GETNUMELEM(
    geometry IN SDO_GEOMETRY
) RETURN NUMBER;
```

Description

Returns the number of elements in the input geometry.

Parameters

geometry

Geometry for which to return the number of elements.

Usage Notes

None.

Examples

The following example returns the number of elements for each geometry in the SHAPE column of the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT c.name, SDO_UTIL.GETNUMELEM(c.shape)
   FROM cola_markets c;
```

NAME	SDO_UTIL.GETNUMELEM(C.SHAPE)
cola_a	1
cola_b	1
cola_c	1
cola_d	1

Related Topics

- [SDO_UTIL.GETNUMVERTICES](#)

SDO_UTIL.GETNUMVERTICES

Format

```
SDO_UTIL.GETNUMVERTICES(  
    geometry IN SDO_GEOMETRY  
    ) RETURN NUMBER;
```

Description

Returns the number of vertices in the input geometry.

Parameters

geometry

Geometry for which to return the number of vertices.

Usage Notes

None.

Examples

The following example returns the number of vertices for each geometry in the SHAPE column of the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT c.name, SDO_UTIL.GETNUMVERTICES(c.shape)  
    FROM cola_markets c;
```

NAME	SDO_UTIL.GETNUMVERTICES(C.SHAPE)
cola_a	2
cola_b	5
cola_c	5
cola_d	3

Related Topics

- [SDO_UTIL.GETVERTICES](#)
- [SDO_UTIL.GETNUMELEM](#)

SDO_UTIL.GETVERTICES

Format

```
SDO_UTIL.GETVERTICES(
    geometry IN SDO_GEOMETRY
) RETURN VERTEX_SET_TYPE;
```

Description

Returns the coordinates of the vertices of the input geometry.

Parameters

geometry

Geometry for which to return the coordinates of the vertices.

Usage Notes

This function returns an object of VERTEX_SET_TYPE, which consists of a table of objects of VERTEX_TYPE. Oracle Spatial defines the type VERTEX_SET_TYPE as:

```
CREATE TYPE vertex_set_type as TABLE OF vertex_type;
```

Oracle Spatial defines the object type VERTEX_TYPE as:

```
CREATE TYPE vertex_type AS OBJECT
(x NUMBER,
 y NUMBER,
 z NUMBER,
 w NUMBER,
 id NUMBER);
```

This function can be useful in finding a vertex that is causing a geometry to be invalid. For example, if you have identified a geometry as invalid by using the [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#) function or the [SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT](#) procedure (both of which are documented in [Chapter 15](#)), you can use the GETVERTICES function to view the vertices in tabular format.

Examples

The following example returns the X and Y coordinates and ID values of the vertices of the geometries in the SHAPE column of the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1](#).)

```
SELECT c.mkt_id, c.name, t.X, t.Y, t.id
FROM cola_markets c,
TABLE(SDO_UTIL.GETVERTICES(c.shape)) t
ORDER BY c.mkt_id, t.id;
```

MKT_ID	NAME	X	Y	ID
1	cola_a	1	1	1
1	cola_a	5	7	2
2	cola_b	5	1	1
2	cola_b	8	1	2

2	cola_b	8	6	3
2	cola_b	5	7	4
2	cola_b	5	1	5
3	cola_c	3	3	1
3	cola_c	6	3	2
3	cola_c	6	5	3
3	cola_c	4	5	4
3	cola_c	3	3	5
4	cola_d	8	7	1
4	cola_d	10	9	2
4	cola_d	8	11	3

15 rows selected.

Related Topics

- [SDO_UTIL.EXTRACT](#)
- [SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT](#)
- [SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT](#)
- [SDO_UTIL.GETNUMVERTICES](#)

SDO_UTIL.INITIALIZE_INDEXES_FOR_TTS

Format

```
SDO_UTIL.INITIALIZE_INDEX_FOR_TTS;
```

Description

Initializes all spatial indexes in a tablespace that was transported to another database.

Parameters

None.

Usage Notes

This procedure is part of the support for using the Oracle transportable tablespace feature with tablespaces that contain any spatial indexes. Use this procedure after the import operation is completed; each user that has a spatial index in the tablespace must call the procedure. You must also have called the [SDO_UTIL.PREPARE_FOR_TTS](#) procedure just before performing the export operation. See the Usage Notes for the [SDO_UTIL.PREPARE_FOR_TTS](#) procedure for more information about using the transportable tablespace feature with spatial data.

For detailed information about transportable tablespaces and transporting tablespaces to other databases, see *Oracle Database Administrator's Guide*.

Examples

The following example initializes all spatial indexes in a tablespace that was transported to another database.

```
CALL SDO_UTIL.INITIALIZE_INDEX_FOR_TTS;
```

Related Topics

- [SDO_UTIL.PREPARE_FOR_TTS](#)

SDO_UTIL.POINT_AT_BEARING

Format

```
SDO_UTIL.POINT_AT_BEARING(
  start_point IN SDO_GEOMETRY,
  bearing     IN NUMBER,
  distance    IN NUMBER
) RETURN SDO_GEOMETRY;
```

Description

Returns a point geometry that is at the specified distance and bearing from the start point.

Parameters

start_point

Point geometry object from which to compute the distance at the specified bearing, to locate the desired point. The point geometry must be based on a geodetic coordinate system.

bearing

Number of radians, measured clockwise from North. Must be in the range of either $-pi$ to pi or 0 to $2*pi$. (Either convention on ranges will work).

distance

Number of meters from `start_point` and along the initial bearing direction to the computed destination point. Must be less than one-half the circumference of the Earth.

Usage Notes

The input point geometry must be based on a geodetic coordinate system. If it is based on a non-geodetic coordinate system, this function returns a null value.

To convert decimal degrees to radians or nonmetric distances to meters, you can use the [SDO_UTIL.CONVERT_UNIT](#) function.

Examples

The following example returns the point 100 kilometers at a bearing of 1 radian from the point with the longitude and latitude coordinates (-72, 43).

```
SELECT SDO_UTIL.POINT_AT_BEARING(
  SDO_GEOMETRY(2001, 8307,
    SDO_POINT_TYPE(-72, 43, NULL), NULL, NULL),
  1, -- 1 radian (57.296 degrees clockwise from North)
  100000 -- 100 kilometers
) FROM DUAL;

SDO_UTIL.POINT_AT_BEARING(SDO_GEOMETRY(2001,8307,SDO_POINT_TYPE(-72,43,NULL),NUL
-----
SDO_GEOMETRY(2001, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 1), SDO_ORDINATE_ARRAY(
-70.957053, 43.4811935))
```


Related Topics

- [SDO_UTIL.CONVERT_UNIT](#)

SDO_UTIL.POLYGONTOLINE

Format

```
SDO_UTIL.POLYGONTOLINE(
    geometry IN SDO_GEOMETRY
) RETURN SDO_GEOMETRY;
```

Description

Converts all polygon-type elements in a geometry to line-type elements, and sets the SDO_GTYPE value accordingly.

Parameters

geometry
Geometry to convert.

Usage Notes

The order of the vertices of each resulting line-type element is the same as in the associated polygon-type element, and the start and end points of each line-type segment are the same point.

If the input geometry is a line, it is returned.

Examples

The following example converts the input polygon geometry, which is the same geometry as `cola_b` (see [Figure 2–1](#) and [Example 2–1](#) in [Section 2.1](#)), to a line string geometry. In the returned geometry, the SDO_GTYPE value (2002) indicates a two-dimensional LINE geometry, and the SDO_ETYPE value (2) and SDO_INTERPRETATION value (1) in the SDO_ELEM_INFO array indicate a line string whose vertices are connected by straight line segments.

```
SELECT SDO_UTIL.POLYGONTOLINE(
    SDO_GEOMETRY(
        2003, -- two-dimensional polygon
        NULL,
        NULL,
        SDO_ELEM_INFO_ARRAY(1,1003,1), -- one polygon (exterior polygon ring)
        SDO_ORDINATE_ARRAY(5,1, 8,1, 8,6, 5,7, 5,1)
    )
) FROM DUAL;

SDO_UTIL.POLYGONTOLINE(SDO_GEOMETRY(2003, --TWO-DIMENSIONALPOLYGONNULL, NULL, SDO_E
-----
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
5, 1, 8, 1, 8, 6, 5, 7, 5, 1))
```

Related Topics

None.

SDO_UTIL.PREPARE_FOR_TTS

Format

```
SDO_UTIL.PREPARE_FOR_TTS(  
    table_space IN VARCHAR2);
```

Description

Prepares a tablespace to be transported to another database, so that spatial indexes will be preserved during the transport operation.

Parameters

table_space
Tablespace to be transported.

Usage Notes

Before Oracle Database 10g release 1 (10.1), the Oracle transportable tablespace feature could not be used with tablespaces that contained any spatial indexes. Effective with Oracle Database 10g release 1 (10.1), you can transport tablespaces that contain spatial indexes; however, you must call the `PREPARE_FOR_TTS` procedure just before you perform the export operation, and you must call it for each user that has a spatial index in the specified tablespace.

Transportable tablespaces containing spatial indexes are not supported across endian format platforms (big-endian to little-endian, or little-endian to big-endian).

After the export operation is complete, you must call the [SDO_UTIL.INITIALIZE_INDEXES_FOR_TTS](#) procedure to initialize all spatial indexes in the transported tablespace.

For detailed information about transportable tablespaces and transporting tablespaces to other databases, see *Oracle Database Administrator's Guide*.

Examples

The following example prepares a tablespace named TS1 to be transported to another database.

```
CALL SDO_UTIL.PREPARE_FOR_TTS('TS1');
```

Related Topics

- [SDO_UTIL.INITIALIZE_INDEXES_FOR_TTS](#)

SDO_UTIL.RECTIFY_GEOMETRY

Format

```
SDO_UTIL.RECTIFY_GEOMETRY(  
    geometry IN SDO_GEOMETRY,  
    tolerance IN NUMBER  
    ) RETURN SDO_GEOMETRY;
```

Description

Fixes certain problems with the input geometry, and returns a valid geometry.

Parameters

geometry

Geometry to be checked for problems that can be fixed.

tolerance

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

This function checks for the following problems that can make a geometry invalid, and fixes the problems in the returned geometry:

- Duplicate vertices
- Polygon boundary intersecting itself
- Incorrect orientation of exterior or interior rings (or both) of a polygon

If the input geometry has any other problem that makes it invalid, the function raises an exception.

If the input geometry is valid, the function returns a geometry that is identical to the input geometry.

This function is used internally by the [SDO_UTIL.SIMPLIFY](#) function as part of the geometry simplification process.

Examples

The following example checks the `cola_b` geometry to see if it has problems that can be fixed. (In this case, the geometry is valid, so the input geometry is returned. The example uses the definitions and data from [Section 2.1](#).)

```
SELECT SDO_UTIL.RECTIFY_GEOMETRY(shape, 0.005)  
    FROM COLA_MARKETS c WHERE c.name = 'cola_b';  
  
SDO_UTIL.RECTIFY_GEOMETRY(SHAPE,0.005)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z),  
-----  
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR  
AY(5, 1, 8, 1, 8, 6, 5, 7, 5, 1))
```

Related Topics

[SDO_UTIL.RECTIFY_GEOMETRY](#)

SDO_UTIL.REMOVE_DUPLICATE_VERTICES

Format

```
SDO_UTIL.REMOVE_DUPLICATE_VERTICES
    geometry IN SDO_GEOMETRY,
    tolerance IN NUMBER
    ) RETURN SDO_GEOMETRY;
```

Description

Removes duplicate (redundant) vertices from a geometry.

Parameters

geometry

Geometry from which to remove duplicate vertices.

tolerance

Tolerance value (see [Section 1.5.5](#)).

Usage Notes

When two consecutive vertices in a geometry are the same or within the tolerance value associated with the geometry, Spatial considers the geometry to be invalid. The Spatial geometry validation functions return the error ORA-13356 in these cases. You can use the REMOVE_DUPLICATE_VERTICES function to change such invalid geometries into valid geometries.

If the input geometry does not contain any duplicate vertices, it is returned.

Examples

The following example removes a duplicate vertex from the input geometry, which is the same geometry as cola_b (see [Figure 2-1](#) and [Example 2-1](#) in [Section 2.1](#)) except that it has been deliberately made invalid by adding a third vertex that is the same point as the second vertex (8,1).

```
SELECT SDO_UTIL.REMOVE_DUPLICATE_VERTICES (
    SDO_GEOMETRY(
        2003, -- two-dimensional polygon
        NULL,
        NULL,
        SDO_ELEM_INFO_ARRAY(1,1003,1), -- one polygon (exterior polygon ring)
        SDO_ORDINATE_ARRAY(5,1, 8,1, 8,1, 8,6, 5,7, 5,1) -- 2nd and 3rd points
        -- are duplicates.
    ),
    0.005 -- tolerance value
) FROM DUAL;

SDO_UTIL.REMOVE_DUPLICATE_VERTICES(SDO_GEOMETRY(2003,--TWO-DIMENSIONALPOLYGONNULL
-----
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(5, 1, 8, 1, 8, 6, 5, 7, 5, 1))
```

Related Topics

None.

SDO_UTIL.REVERSE_LINestring

Format

```
SDO_UTIL.REVERSE_LINestring(  
    geometry IN SDO_GEOMETRY  
    ) RETURN SDO_GEOMETRY;
```

Description

Returns a line string geometry with the vertices of the input geometry in reverse order.

Parameters

geometry

Line string geometry whose vertices are to be reversed in the output geometry. The SDO_GTYPE value of the input geometry must be 2002. ([Section 2.2.1](#) explains SDO_GTYPE values.)

Usage Notes

Because the SDO_GTYPE value of the input geometry must be 2002, this function cannot be used to reverse LRS geometries. To reverse an LRS geometry, use the [SDO_LRS.REVERSE_GEOMETRY](#) function, which is described in [Chapter 16](#).

Examples

The following example returns a line string geometry that reverses the vertices of the input geometry.

```
SELECT SDO_UTIL.REVERSE_LINestring(  
    SDO_GEOMETRY(2002, 8307, NULL, SDO_ELEM_INFO_ARRAY(1,2,1),  
    SDO_ORDINATE_ARRAY(-72,43, -71.5,43.5, -71,42, -70,40))  
    ) FROM DUAL;
```

```
SDO_UTIL.REVERSE_LINestring(SDO_GEOMETRY(2002,8307,NULL,SDO_ELEM_INFO_ARRAY(1,2,  
-----  
SDO_GEOMETRY(2002, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(  
-70, 40, -71, 42, -71.5, 43.5, -72, 43))
```

Related Topics

- [SDO_LRS.REVERSE_GEOMETRY](#) (in [Chapter 16](#))

SDO_UTIL.SIMPLIFY

Format

```
SDO_UTIL.SIMPLIFY(  
    geometry IN SDO_GEOMETRY,  
    threshold IN NUMBER  
    tolerance IN NUMBER DEFAULT 0.0000005  
) RETURN SDO_GEOMETRY;
```

Description

Simplifies the input geometry, based on a threshold value, using the Douglas-Peucker algorithm.

Parameters

geometry

Geometry to be simplified.

threshold

Threshold value to be used for the geometry simplification. Should be a positive number. (Zero causes the input geometry to be returned.) If the input geometry is geodetic, the value is the number of meters; if the input geometry is non-geodetic, the value is the number of units associated with the data.

As the threshold value is decreased, the returned geometry is likely to be closer to the input geometry; as the threshold value is increased, fewer points are likely to be in the returned geometry. See the Usage Notes for more information.

tolerance

Tolerance value (see [Section 1.5.5](#)). Must not be greater than `threshold`; and for better performance, should not be the same as `threshold`. If you do not specify a value, the default value is 0.0000005.

Usage Notes

This function also convert arcs to line stings, eliminates duplicate vertices, and corrects many overlapping edge polygon problems. The reason this function sometimes fixes problems is that it internally calls the [SDO_UTIL.RECTIFY_GEOMETRY](#) function at the end of the simplification process to ensure that a valid geometry is returned.

This function is useful when you want a geometry with less fine resolution than the original geometry. For example, if the display resolution cannot show the hundreds or thousands of turns in the course of a river or in a political boundary, better performance might result if the geometry were simplified to show only the major turns.

If you use this function with geometries that have more than two dimensions, only the first two dimensions are used in processing the query, and only the first two dimensions in the returned geometry are to be considered valid and meaningful. For example, the measure values in a returned LRS geometry will probably not reflect actual measures in that geometry. In this case, depending on your application needs,

you might have several options after the simplification operation, such as ignoring the new measure values or redefining the new LRS geometry to reset the measure values.

This function uses the Douglas-Peucker algorithm, which is explained in several cartography textbooks and reference documents. (In some explanations, the term *tolerance* is used instead of *threshold*; however, this is different from the Oracle Spatial meaning of tolerance.)

The returned geometry can be a polygon, line, or point, depending on the geometry definition and the threshold value. The following considerations apply:

- A polygon can simplify to a line or a point and a line can simplify to a point, if the threshold value associated with the geometry is sufficiently large. For example, a thin rectangle will simplify to a line if the distance between the two parallel long sides is less than the threshold value, and a line will simplify to a point if the distance between the start and end points is less than the threshold value.
- In a polygon with a hole, if the exterior ring or the interior ring (the hole) simplifies to a line or a point, the interior ring disappears from (is not included in) the resulting geometry.
- Topological characteristics of the input geometry might not be maintained after simplification. For a collection geometry, the number of elements might increase, to prevent overlapping of individual elements. In all cases, this function will not return an invalid geometry.

Examples

The following example simplifies the road shown in [Figure 7–20](#) in [Section 7.7](#). Because the threshold value (6) is fairly large given the input geometry, the resulting LRS line string has only three points: the start and end points, and (12, 4,12). The measure values in the returned geometry are not meaningful, because this function considers only two dimensions.

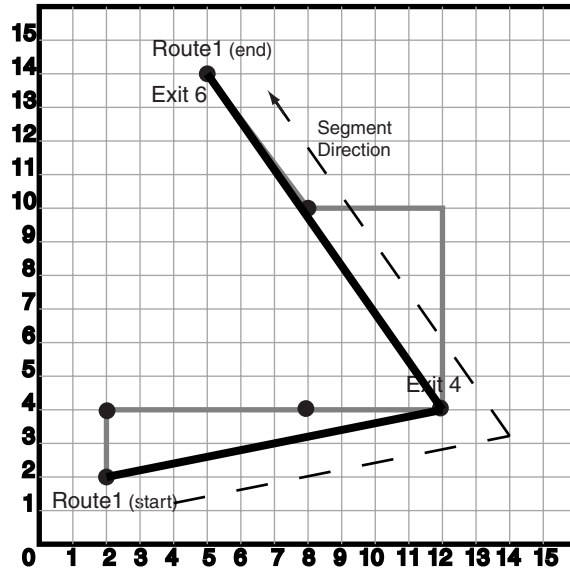
```
SELECT SDO_UTIL.SIMPLIFY(
  SDO_GEOMETRY(
    3302, -- line string, 3 dimensions (X,Y,M), 3rd is linear ref. dimension
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1), -- one line string, straight segments
    SDO_ORDINATE_ARRAY(
      2,2,0, -- Starting point - Exit1; 0 is measure from start.
      2,4,2, -- Exit2; 2 is measure from start.
      8,4,8, -- Exit3; 8 is measure from start.
      12,4,12, -- Exit4; 12 is measure from start.
      12,10,NULL, -- Not an exit; measure automatically calculated and filled.
      8,10,22, -- Exit5; 22 is measure from start.
      5,14,27) -- Ending point (Exit6); 27 is measure from start.
    ),
  6, -- threshold value for geometry simplification
  0.5 -- tolerance
) FROM DUAL;

SDO_UTIL.SIMPLIFY(SDO_GEOMETRY(3302,--LINESTRING,3DIMENSIONS(X,Y,M),3RDISLINEARR
-----
SDO_GEOMETRY(3302, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 0, 12, 4, 12, 5, 14, 27))
```

[Figure 20–1](#) shows the result of this example. In [Figure 20–1](#), the thick solid black line is the resulting geometry, the thin solid light line between the start and end points is

the input geometry, and the thin dashed line with the arrowhead at the end shows the direction of the segment.

Figure 20–1 Simplification of a Geometry



Related Topics

[SDO_UTIL.RECTIFY_GEOMETRY](#)

SDO_UTIL.TO_GMLGEOMETRY

Format

```
SDO_UTIL.TO_GMLGEOMETRY(  
    thegeom IN SDO_GEOMETRY  
    ) RETURN CLOB;
```

Description

Converts a Spatial geometry object to a geography markup language (GML 2.0) fragment based on the geometry types defined in the Open GIS `geometry.xsd` schema document.

Parameters

thegeom

Geometry for which to return the GML fragment.

Usage Notes

This function does not convert circles, geometries containing any circular arcs, LRS geometries, or geometries with an SDO_ETYPE value of 0 (type 0 elements); it returns an empty CLOB in these cases.

This function converts the input geometry to a GML fragment based on some GML geometry types defined in the Open GIS Implementation Specification.

The input geometry must have a 4-digit SDO_GTYPE value.

Polygons must be defined using the conventions for Oracle9i and later releases of Spatial. That is, the outer boundary is stored first (with ETYPE=1003) followed by zero or more inner boundary elements (ETYPE=2003). For a polygon with holes, the outer boundary must be stored first in the SDO_ORDINATES definition, followed by coordinates of the inner boundaries.

LRS geometries must be converted to standard geometries (using the [SDO_LRS.CONVERT_TO_STD_GEOM](#) or [SDO_LRS.CONVERT_TO_STD_LAYER](#) function) before being passed to the TO_GMLGEOMETRY function. (See the Examples section for an example that uses CONVERT_TO_STD_GEOM with the TO_GMLGEOMETRY function.)

Any circular arcs or circles must be densified (using the [SDO_GEOM.SDO_ARC_DENSIFY](#) function) or represented as polygons (using the [SDO_GEOM.SDO_BUFFER](#) function) before being passed to the TO_GMLGEOMETRY function. (See the Examples section for an example that uses SDO_ARC_DENSIFY with the TO_GMLGEOMETRY function.)

Label points are discarded. That is, if a geometry has a value for the SDO_POINT field and values in SDO_ELEM_INFO and SDO_ORDINATES, the SDO_POINT is not output in the GML fragment.

The SDO_SRID value is output in the form `srsName="SDO:<srid>"`. For example, `"SDO:8307"` indicates SDO_SRID 8307, and `"SDO:"` indicates a null SDO_SRID value. No checks are made for the validity or consistency of the SDO_SRID value. For example, the value is not checked to see if it exists in the MDSYS.CS_SRS table or if it

conflicts with the SRID value for the layer in the USER_SDO_GEOM_METADATA view.

Coordinates are always output using the `<coordinates>` tag and `decimal='.'`, `cs=','` (that is, with the comma as the coordinate separator), and `ts=' '` (that is, with a space as the tuple separator), even if the NLS_NUMERIC_CHARACTERS setting has `','` (comma) as the decimal character.

The GML output is not formatted; there are no line breaks or indentation of tags. To see the contents of the returned CLOB in SQL*Plus, use the TO_CHAR() function or set the SQL*Plus parameter LONG to a suitable value (for example, SET LONG 40000). To get formatted GML output or to use the return value of TO_GMLGEOMETRY in SQLX or Oracle XML DB functions such as XMLELEMENT, use the XMLTYPE(clobval CLOB) constructor.

Examples

The following example returns the GML fragment for the cola_b geometry in the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1.](#))

```
-- Convert cola_b geometry to GML fragment.
SELECT TO_CHAR(SDO_UTIL.TO_GMLGEOMETRY(shape)) AS GmlGeometry
  FROM COLA_MARKETS c WHERE c.name = 'cola_b';

GMLGEOMETRY
-----
<gml:Polygon srsName="SDO:" xmlns:gml="http://www.opengis.net/gml"><gml:outerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">5,1 8,1 8,6 5,7 5,1 </gml:coordinates></gml:LinearRing></gml:outerBoundaryIs></gml:Polygon>
```

The following example returns the GML fragment for the arc densification of the cola_d geometry in the COLA_MARKETS table. (The example uses the definitions and data from [Section 2.1.](#))

```
SET LONG 40000
SELECT XMLTYPE(SDO_UTIL.TO_GMLGEOMETRY(
  SDO_GEOM.SDO_ARC_DENSIFY(c.shape, m.diminfo, 'arc_tolerance=0.05')))
  AS GmlGeometry FROM cola_markets c, user_sdo_geom_metadata m
  WHERE m.table_name = 'COLA_MARKETS' AND m.column_name = 'SHAPE'
  AND c.name = 'cola_d';

GMLGEOMETRY
-----
<gml:Polygon srsName="SDO:" xmlns:gml="http://www.opengis.net/gml"><gml:outerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">8,7 8.7653668 6473018,7.15224093497743 9.4142135623731,7.58578643762691 9.84775906502257,8.234 63313526982 10,9 9.84775906502257,9.76536686473018 9.4142135623731,10.4142135623 731 8.76536686473018,10.8477590650226 8,11 7.23463313526982,10.8477590650226 6.5 8578643762691,10.4142135623731 6.15224093497743,9.76536686473018 6,9 6.152240934 97743,8.23463313526982 6.58578643762691,7.5857864376269 7.23463313526982,7.15224 093497743 8,7 </gml:coordinates></gml:LinearRing></gml:outerBoundaryIs></gml:Pol ygon>
```

The following example converts an LRS geometry to a standard geometry and returns the GML fragment for the geometry. (The example uses the definitions and data from [Section 7.7.](#))

```
SET LONG 40000
-- Convert LRS geometry to standard geometry before using TO_GMLGEOMETRY.
SELECT XMLTYPE(SDO_UTIL.TO_GMLGEOMETRY(
  SDO_LRS.CONVERT_TO_STD_GEOM(route_geometry)))
```

```
AS GmlGeometry FROM lrs_routes a WHERE a.route_id = 1;
```

```
GMLGEOMETRY
```

```
-----
<gml:LineString srsName="SDO:" xmlns:gml="http://www.opengis.net/gml">
  <gml:coordinates decimal="." cs="," ts=" ">2,2 2,4 8,4 12,4 12,10 8,10 5,14 </
gml:coordinates>
</gml:LineString>
```

The following examples return GML fragments for a variety of geometry types.

```
-- Point geometry with coordinates in SDO_ORDINATES. Note the
-- coordinates in the GML are (10,10) and the values in the
-- SDO_POINT field are discarded.
```

```
SELECT TO_CHAR(
  SDO_UTIL.TO_GMLGEOMETRY(sdo_geometry(2001, 8307,
    sdo_point_type(-80, 70, null),
    sdo_elem_info_array(1,1,1), sdo_ordinate_array(10, 10)))
)
AS GmlGeometry FROM DUAL;
```

```
GMLGEOMETRY
```

```
-----
<gml:Point srsName="SDO:8307" xmlns:gml="http://www.opengis.net/gml"><gml:coordi
nates decimal="." cs="," ts=" ">10,10 </gml:coordinates></gml:Point>
```

```
-- LRS geometry. An Empty CLOB is returned.
```

```
SELECT SDO_UTIL.TO_GMLGEOMETRY(
  sdo_geometry(2306, 8307, null,
    sdo_elem_info_array(1,1003,1, 13, 1003, 1, 23, 1003, 3),
    sdo_ordinate_array(10.10,10.20, 20.50, 20.10, 30.30, 30.30, 40.10,
      40.10, 30.50, 30.20, 10.10, 10.20,
      5, 5, 5, 6, 6, 6, 6, 5, 5, 5, 7, 7, 8, 8 ))
) AS GmlGeometry FROM DUAL;
```

```
GMLGEOMETRY
```

```
-- Rectangle (geodetic)
```

```
SELECT TO_CHAR(
  SDO_UTIL.TO_GMLGEOMETRY(sdo_geometry(2003, 8307, null,
    sdo_elem_info_array(1,1003,5),
    sdo_ordinate_array(10.10,10.10, 20.10, 20.10 )))
)
AS GmlGeometry FROM DUAL;
```

```
GMLGEOMETRY
```

```
-----
<gml:Box srsName="SDO:8307" xmlns:gml="http://www.opengis.net/gml"><gml:coordina
tes decimal="." cs="," ts=" ">10.1,10.1 20.1,20.1 </gml:coordinates></gml:Box>
```

```
-- Polygon with holes
```

```
SELECT TO_CHAR(
  SDO_UTIL.TO_GMLGEOMETRY(sdo_geometry(2003, 262152, null,
    sdo_elem_info_array(1,1003,3, 5, 2003, 1, 13, 2003, 1),
    sdo_ordinate_array(10.10,10.20, 40.50, 41.10, 30.30, 30.30, 30.30,
      40.10, 40.10, 40.10, 30.30, 30.30, 5, 5, 5, 6, 6, 6, 6, 5, 5, 5 )))
)
```

```
AS GmlGeometry FROM DUAL;
```

```
GMLGEOMETRY
```

```
-----
<gml:Polygon srsName="SDO:262152" xmlns:gml="http://www.opengis.net/gml"><gml:outerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">10.1,10.2, 40.5,10.2, 40.5,41.1, 10.1,41.1, 10.1,10.2 </gml:coordinates></gml:LinearRing></gml:outerBoundaryIs><gml:innerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">30.3,30.3 30.3,40.1 40.1,40.1 30.3,30.3 </gml:coordinates></gml:LinearRing></gml:innerBoundaryIs><gml:innerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">5,5 5,6 6,6 6,5 5,5 </gml:coordinates></gml:LinearRing></gml:innerBoundaryIs></gml:Polygon>
```

```
-- Creating an XMLTYPE from the GML fragment. Also useful for "pretty
-- printing" the GML output.
```

```
SET LONG 40000
```

```
SELECT XMLTYPE(
```

```
    SDO_UTIL.TO_GMLGEOMETRY(sdo_geometry(2003, 262152, null,
    sdo_elem_info_array(1,1003,1, 11, 2003, 1, 21, 2003, 1),
    sdo_ordinate_array(10.10,10.20, 40.50,10.2, 40.5,41.10, 10.1,41.1,
    10.10, 10.20, 30.30,30.30, 30.30, 40.10, 40.10, 40.10, 40.10, 30.30,
    30.30, 30.30, 5, 5, 5, 6, 6, 6, 6, 5, 5, 5 )))
```

```
)
```

```
AS GmlGeometry FROM DUAL;
```

```
GMLGEOMETRY
```

```
-----
<gml:Polygon srsName="SDO:262152" xmlns:gml="http://www.opengis.net/gml"><gml:outerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">10.1,10.2 40.5,10.2 40.5,41.1 10.1,41.1 10.1,10.2 </gml:coordinates></gml:LinearRing></gml:outerBoundaryIs><gml:innerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">30.3,30.3 30.3,40.1 40.1,40.1 40.1,30.3 30.3,30.3 </gml:coordinates></gml:LinearRing></gml:innerBoundaryIs><gml:innerBoundaryIs><gml:LinearRing><gml:coordinates decimal="." cs="," ts=" ">5,5 5,6 6,6 6,5 5,5 </gml:coordinates></gml:LinearRing></gml:innerBoundaryIs></gml:Polygon>
```

The following example uses the TO_GMLGEOMETRY function with the Oracle XML DB XMLTYPE data type and the XMLELEMENT and XMLFOREST functions.

```
SELECT xmlelement("State", xmlattributes(
    'http://www.opengis.net/gml' as "xmlns:gml"),
    xmlforest(state as "Name", totpop as "Population",
    xmltype(sdo_util.to_gmlgeometry(geom)) as "gml:geometryProperty"))
AS theXMLElements FROM states WHERE state_abrv in ('DE', 'UT');
```

```
THEXMLEMENTS
```

```
-----
<State xmlns:gml="http://www.opengis.net/gml">
  <Name>Delaware</Name>
  <Population>666168</Population>
  <gml:geometryProperty>
    <gml:Polygon srsName="SDO:" xmlns:gml="http://www.opengis.net/gml">
      <gml:outerBoundaryIs>
        <gml:LinearRing>
          <gml:coordinates decimal="." cs="," ts=" ">-75.788704,39.721699 -75.78
8704,39.6479 -75.767014,39.377106 -75.76033,39.296497 -75.756294,39.24585 -75.74
8016,39.143196 -75.722961,38.829895 -75.707695,38.635166 -75.701912,38.560619 -7
5.693871,38.460011 -75.500336,38.454002 -75.341614,38.451855 -75.049339,38.45165
3 -75.053841,38.538429 -75.06015,38.605465 -75.063263,38.611275 -75.065308,38.62
949 -75.065887,38.660919 -75.078697,38.732403 -75.082527,38.772045 -75.091667,38
```

```

.801208 -75.094185,38.803699 -75.097572,38.802986 -75.094116,38.793579 -75.09926
6,38.78756 -75.123619,38.781784 -75.137962,38.782703 -75.18692,38.803772 -75.215
019,38.831547 -75.23735,38.849014 -75.260498,38.875 -75.305908,38.914673 -75.316
399,38.930309 -75.317284,38.93676 -75.312851,38.945576 -75.312859,38.945618 -75.
31205,38.967804 -75.31778,38.986012 -75.341431,39.021233 -75.369606,39.041359 -7
5.389229,39.051422 -75.40181,39.06702 -75.401306,39.097713 -75.411369,39.148029
-75.407845,39.175201 -75.396271,39.187778 -75.39225,39.203377 -75.40181,39.23104
9 -75.402817,39.253189 -75.409355,39.264759 -75.434006,39.290424 -75.439041,39.3
13065 -75.453125,39.317093 -75.457657,39.326653 -75.469231,39.330677 -75.486336,
39.341743 -75.494888,39.354324 -75.504448,39.357346 -75.51284,39.366291 -75.5129
24,39.366482 -75.523773,39.392052 -75.538651,39.415707 -75.56749,39.436436 -75.5
9137,39.463696 -75.592941,39.471806 -75.590019,39.488026 -75.587311,39.496136 -7
5.5774,39.508076 -75.554192,39.506947 -75.528442,39.498005 -75.530373,39.510303
-75.527145,39.531326 -75.52803,39.535168 -75.53437,39.540592 -75.519386,39.55528
6 -75.512291,39.567505 -75.515587,39.580639 -75.528046,39.584 -75.538269,39.5935
67 -75.554016,39.601727 -75.560143,39.622578 -75.556602,39.6348 -75.549599,39.63
7699 -75.542397,39.645901 -75.535507,39.647099 -75.514999,39.668499 -75.507523,3
9.69685 -75.496597,39.701302 -75.488914,39.714722 -75.477997,39.714901 -75.47550
2,39.733501 -75.467972,39.746975 -75.463707,39.761101 -75.448494,39.773857 -75.4
38301,39.783298 -75.405701,39.796101 -75.415405,39.801678 -75.454102,39.820202 -
75.499199,39.833199 -75.539703,39.8381 -75.5802,39.838417 -75.594017,39.837345 -
75.596107,39.837044 -75.639488,39.82893 -75.680145,39.813839 -75.71096,39.796352
-75.739716,39.772881 -75.760689,39.74712 -75.774101,39.721699 -75.788704,39.721
699 </gml:coordinates>
    </gml:LinearRing>
  </gml:outerBoundaryIs>
</gml:Polygon>
</gml:geometryProperty>
</State>

<State xmlns:gml="http://www.opengis.net/gml">
  <Name>Utah</Name>
  <Population>1722850</Population>
  <gml:geometryProperty>
    <gml:Polygon srsName="SDO:" xmlns:gml="http://www.opengis.net/gml">
      <gml:outerBoundaryIs>
        <gml:LinearRing>
          <gml:coordinates decimal="." cs="," ts=" ">-114.040871,41.993805 -114.
038803,41.884899 -114.041306,41 -114.04586,40.116997 -114.046295,39.906101 -114.
046898,39.542801 -114.049026,38.67741 -114.049339,38.572968 -114.049095,38.14864
-114.0476,37.80946 -114.05098,37.746284 -114.051666,37.604805 -114.052025,37.10
3989 -114.049797,37.000423 -113.484375,37 -112.898598,37.000401 -112.539604,37.0
00683 -112,37.000977 -111.412048,37.001514 -111.133018,37.00079 -110.75,37.00320
1 -110.5,37.004265 -110.469505,36.998001 -110,36.997967 -109.044571,36.999088 -1
09.045143,37.375 -109.042824,37.484692 -109.040848,37.881176 -109.041405,38.1530
27 -109.041107,38.1647 -109.059402,38.275501 -109.059296,38.5 -109.058868,38.719
906 -109.051765,39 -109.050095,39.366699 -109.050697,39.4977 -109.050499,39.6605
-109.050156,40.222694 -109.047577,40.653641 -109.0494,41.000702 -109.2313,41.00
2102 -109.534233,40.998184 -110,40.997398 -110.047768,40.997696 -110.5,40.994801
-111.045982,40.998013 -111.045815,41.251774 -111.045097,41.579899 -111.045944,4
2.001633 -111.506493,41.999588 -112.108742,41.997677 -112.16317,41.996784 -112.1
72562,41.996643 -112.192184,42.001244 -113,41.998314 -113.875,41.988091 -114.040
871,41.993805 </gml:coordinates>
        </gml:LinearRing>
      </gml:outerBoundaryIs>
    </gml:Polygon>
  </gml:geometryProperty>
</State>

```

Related Topics

None.

SDO_UTIL.TO_WKBGEOMETRY

Format

```
SDO_UTIL.TO_WKBGEOMETRY(
    geometry IN SDO_GEOMETRY
) RETURN BLOB;
```

Description

Converts a Spatial geometry object to the well-known binary (WKB) format.

Parameters

geometry
SDO_GEOMETRY object to be converted to WKB format.

Usage Notes

The input geometry is converted to the well-known binary (WKB) format, as defined by the Open Geospatial Consortium and the International Organization for Standardization (ISO).

This function is patterned after the SQL Multimedia recommendations in *ISO 13249-3, Information technology - Database languages - SQL Multimedia and Application Packages - Part 3: Spatial*.

To convert a geometry in WKB format to an SDO_GEOMETRY object, use the [SDO_UTIL.FROM_WKBGEOMETRY](#) function.

Examples

The following example shows conversion to and from WKB and WKT format, and validation of WKB and WKT geometries. (The example uses the definitions and data from [Section 2.1](#), specifically the `cola_b` geometry from the `COLA_MARKETS` table.)

```
DECLARE
    wkbgeom BLOB;
    wktgeom CLOB;
    val_result VARCHAR2(5);
    geom_result SDO_GEOMETRY;
    geom SDO_GEOMETRY;
BEGIN
    SELECT c.shape INTO geom FROM cola_markets c WHERE c.name = 'cola_b';

    -- To WBT/WKT geometry
    wkbgeom := SDO_UTIL.TO_WKBGEOMETRY(geom);
    wktgeom := SDO_UTIL.TO_WKTGEOMETRY(geom);
    DBMS_OUTPUT.PUT_LINE('To WKT geometry result = ' || TO_CHAR(wktgeom));

    -- From WBT/WKT geometry
    geom_result := SDO_UTIL.FROM_WKBGEOMETRY(wkbgeom);
    geom_result := SDO_UTIL.FROM_WKTGEOMETRY(wktgeom);

    -- Validate WBT/WKT geometry
    val_result := SDO_UTIL.VALIDATE_WKBGEOMETRY(wkbgeom);
    DBMS_OUTPUT.PUT_LINE('WKB validation result = ' || val_result);
```

```
val_result := SDO_UTIL.VALIDATE_WKTGEOMETRY(wktgeom);
DBMS_OUTPUT.PUT_LINE('WKT validation result = ' || val_result);

END;
/
```

```
To WKT geometry result = POLYGON ((5.0 1.0, 8.0 1.0, 8.0 6.0, 5.0 7.0, 5.0 1.0))
WKB validation result = TRUE
WKT validation result = TRUE
```

Related Topics

- [SDO_UTIL.FROM_WKBGEOMETRY](#)
- [SDO_UTIL.FROM_WKTGEOMETRY](#)
- [SDO_UTIL.TO_WKTGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKBGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKTGEOMETRY](#)

SDO_UTIL.TO_WKTGEOMETRY

Format

```
SDO_UTIL.TO_WKTGEOMETRY(
    geometry IN SDO_GEOMETRY
) RETURN CLOB;
```

Description

Converts a Spatial geometry object to the well-known text (WKT) format.

Parameters

geometry

SDO_GEOMETRY object to be converted to WKT format.

Usage Notes

The input geometry is converted to the well-known text (WKT) format, as defined by the Open Geospatial Consortium and the International Organization for Standardization (ISO).

This function is patterned after the SQL Multimedia recommendations in *ISO 13249-3, Information technology - Database languages - SQL Multimedia and Application Packages - Part 3: Spatial*.

To convert a geometry in WKT format to an SDO_GEOMETRY object, use the [SDO_UTIL.FROM_WKTGEOMETRY](#) function.

Examples

The following example shows conversion to and from WKB and WKT format, and validation of WKB and WKT geometries. (The example uses the definitions and data from [Section 2.1](#), specifically the `cola_b` geometry from the `COLA_MARKETS` table.)

```
DECLARE
    wkbgeom BLOB;
    wktgeom CLOB;
    val_result VARCHAR2(5);
    geom_result SDO_GEOMETRY;
    geom SDO_GEOMETRY;
BEGIN
    SELECT c.shape INTO geom FROM cola_markets c WHERE c.name = 'cola_b';

    -- To WBT/WKT geometry
    wkbgeom := SDO_UTIL.TO_WKBGEOMETRY(geom);
    wktgeom := SDO_UTIL.TO_WKTGEOMETRY(geom);
    DBMS_OUTPUT.PUT_LINE('To WKT geometry result = ' || TO_CHAR(wktgeom));

    -- From WBT/WKT geometry
    geom_result := SDO_UTIL.FROM_WKBGEOMETRY(wkbgeom);
    geom_result := SDO_UTIL.FROM_WKTGEOMETRY(wktgeom);

    -- Validate WBT/WKT geometry
    val_result := SDO_UTIL.VALIDATE_WKBGEOMETRY(wkbgeom);
    DBMS_OUTPUT.PUT_LINE('WKB validation result = ' || val_result);
```

```
val_result := SDO_UTIL.VALIDATE_WKTGEOMETRY(wktgeom);  
DBMS_OUTPUT.PUT_LINE('WKT validation result = ' || val_result);  
  
END;  
/
```

```
To WKT geometry result = POLYGON ((5.0 1.0, 8.0 1.0, 8.0 6.0, 5.0 7.0, 5.0 1.0))  
WKB validation result = TRUE  
WKT validation result = TRUE
```

Related Topics

- [SDO_UTIL.FROM_WKBGEOMETRY](#)
- [SDO_UTIL.FROM_WKTGEOMETRY](#)
- [SDO_UTIL.TO_WKBGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKBGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKTGEOMETRY](#)

SDO_UTIL.VALIDATE_WKBGEOMETRY

Format

```
SDO_UTIL.VALIDATE_WKBGEOMETRY(
    geometry IN BLOB
) RETURN VARCHAR2;
```

Description

Validates the input geometry, which is in the standard well-known binary (WKB) format; returns the string `TRUE` if the geometry is valid or `FALSE` if the geometry is not valid.

Parameters

geometry

Geometry in WKB format to be checked for validity.

Usage Notes

To be valid, the input geometry must be in the well-known binary (WKB) format, as defined by the Open Geospatial Consortium and the International Organization for Standardization (ISO).

This function is patterned after the SQL Multimedia recommendations in *ISO 13249-3, Information technology - Database languages - SQL Multimedia and Application Packages - Part 3: Spatial*.

To validate a geometry in the well-known text (WKT) format, use the [SDO_UTIL.VALIDATE_WKTGEOMETRY](#) function.

Examples

The following example shows conversion to and from WKB and WKT format, and validation of WKB and WKT geometries. (The example uses the definitions and data from [Section 2.1](#), specifically the `cola_b` geometry from the `COLA_MARKETS` table.)

```
DECLARE
    wkbgeom BLOB;
    wktgeom CLOB;
    val_result VARCHAR2(5);
    geom_result SDO_GEOMETRY;
    geom SDO_GEOMETRY;
BEGIN
    SELECT c.shape INTO geom FROM cola_markets c WHERE c.name = 'cola_b';

    -- To WBT/WKT geometry
    wkbgeom := SDO_UTIL.TO_WKBGEOMETRY(geom);
    wktgeom := SDO_UTIL.TO_WKTGEOMETRY(geom);
    DBMS_OUTPUT.PUT_LINE('To WKT geometry result = ' || TO_CHAR(wktgeom));

    -- From WBT/WKT geometry
    geom_result := SDO_UTIL.FROM_WKBGEOMETRY(wkbgeom);
    geom_result := SDO_UTIL.FROM_WKTGEOMETRY(wktgeom);

    -- Validate WBT/WKT geometry
```

```
val_result := SDO_UTIL.VALIDATE_WKBGEOMETRY(wkbgeom);
DBMS_OUTPUT.PUT_LINE('WKB validation result = ' || val_result);
val_result := SDO_UTIL.VALIDATE_WKTGEOMETRY(wktgeom);
DBMS_OUTPUT.PUT_LINE('WKT validation result = ' || val_result);

END;
/
```

```
To WKT geometry result = POLYGON ((5.0 1.0, 8.0 1.0, 8.0 6.0, 5.0 7.0, 5.0 1.0))
WKB validation result = TRUE
WKT validation result = TRUE
```

Related Topics

- [SDO_UTIL.FROM_WKBGEOMETRY](#)
- [SDO_UTIL.FROM_WKTGEOMETRY](#)
- [SDO_UTIL.TO_WKBGEOMETRY](#)
- [SDO_UTIL.TO_WKTGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKTGEOMETRY](#)

SDO_UTIL.VALIDATE_WKTGEOMETRY

Format

```
SDO_UTIL.VALIDATE_WKTGEOMETRY(
    geometry IN CLOB
) RETURN VARCHAR2;
```

or

```
SDO_UTIL.VALIDATE_WKTGEOMETRY(
    geometry IN VARCHAR2
) RETURN VARCHAR2;
```

Description

Validates the input geometry, which is of type CLOB or VARCHAR2 and in the standard well-known text (WKT) format; returns the string TRUE if the geometry is valid or FALSE if the geometry is not valid.

Parameters

geometry

Geometry in WKT format to be checked for validity.

Usage Notes

To be valid, the input geometry must be in the well-known text (WKT) format, as defined by the Open Geospatial Consortium and the International Organization for Standardization (ISO).

This function is patterned after the SQL Multimedia recommendations in *ISO 13249-3, Information technology - Database languages - SQL Multimedia and Application Packages - Part 3: Spatial*.

To validate a geometry in the well-known binary (WKB) format, use the [SDO_UTIL.VALIDATE_WKBGEOMETRY](#) function.

Examples

The following example shows conversion to and from WKB and WKT format, and validation of WKB and WKT geometries. (The example uses the definitions and data from [Section 2.1](#), specifically the `cola_b` geometry from the `COLA_MARKETS` table.)

```
DECLARE
    wkbgeom BLOB;
    wktgeom CLOB;
    val_result VARCHAR2(5);
    geom_result SDO_GEOMETRY;
    geom SDO_GEOMETRY;
BEGIN
    SELECT c.shape INTO geom FROM cola_markets c WHERE c.name = 'cola_b';

    -- To WBT/WKT geometry
    wkbgeom := SDO_UTIL.TO_WKBGEOMETRY(geom);
    wktgeom := SDO_UTIL.TO_WKTGEOMETRY(geom);
    DBMS_OUTPUT.PUT_LINE('To WKT geometry result = ' || TO_CHAR(wktgeom));
```

```
-- From WBT/WKT geometry
geom_result := SDO_UTIL.FROM_WKBGEOMETRY(wkbgeom);
geom_result := SDO_UTIL.FROM_WKTGEOMETRY(wktgeom);

-- Validate WBT/WKT geometry
val_result := SDO_UTIL.VALIDATE_WKBGEOMETRY(wkbgeom);
DBMS_OUTPUT.PUT_LINE('WKB validation result = ' || val_result);
val_result := SDO_UTIL.VALIDATE_WKTGEOMETRY(wktgeom);
DBMS_OUTPUT.PUT_LINE('WKT validation result = ' || val_result);

END;
/

To WKT geometry result = POLYGON ((5.0 1.0, 8.0 1.0, 8.0 6.0, 5.0 7.0, 5.0 1.0))
WKB validation result = TRUE
WKT validation result = TRUE
```

Related Topics

- [SDO_UTIL.FROM_WKBGEOMETRY](#)
- [SDO_UTIL.FROM_WKTGEOMETRY](#)
- [SDO_UTIL.TO_WKBGEOMETRY](#)
- [SDO_UTIL.TO_WKTGEOMETRY](#)
- [SDO_UTIL.VALIDATE_WKBGEOMETRY](#)

Part III

Supplementary Information

This document has three parts:

- [Part I](#) provides conceptual and usage information about Oracle Spatial.
- [Part II](#) provides reference information about Oracle Spatial operators, functions, and procedures.
- Part III provides supplementary information (appendixes and a glossary).

Part III contains the following:

- [Appendix A, "Installation, Compatibility, and Upgrade"](#)
- [Appendix B, "Oracle Locator"](#)
- [Appendix C, "Routing Engine"](#)
- [Appendix D, "Complex Spatial Queries: Examples"](#)
- [Glossary](#)

Installation, Compatibility, and Upgrade

If you are upgrading to Oracle Database 10g, Oracle Spatial is automatically upgraded as part of the operation. For information about the upgrade procedure, see *Oracle Database Upgrade Guide*.

If you need to downgrade Spatial to the previous Oracle Database release, follow the instructions for downgrading a database back to the previous Oracle Database release in *Oracle Database Upgrade Guide*.

If you have LRS data in release 8.1.5, 8.1.6, or 8.1.7 format, see [Section A.1](#).

If you use Oracle Spatial GeoRaster, see [Section A.2](#).

A.1 Upgrading LRS Data

If you have linear referencing data (that is, geometries with measure information) in release 8.1.5, 8.1.6, or 8.1.7 format, you must upgrade that data to the format for Spatial releases 9.0.1 and later, as follows:

1. Drop any spatial indexes on the table with the linear referencing data.
2. Find out which dimension of the object has the linear referencing information.

This could be the third or the fourth dimension, depending on the dimensionality of the data. For example, if the data has three dimensions (such as X, Y, and height), the LRS geometry object is 4D, and the LRS dimension in this case is usually 4.

3. Make sure that the data is in the format for release 8.1.6 or later (that is, it has 4-digit SDO_GTYPE values).
4. Update the LRS geometry objects by setting the LRS dimension in the SDO_GTYPE field, as in the following examples.

Example 1: The LRS dimension is 3 for the geometries in the GEOMETRY column of table LRS_DATA. Update the SDO_GTYPE as follows:

```
UPDATE LRS_DATA a SET a.geometry.sdo_gtype = a.geometry.sdo_gtype + 300;
```

Example 2: The LRS dimension is 4 for the geometries in the GEOMETRY column of table LRS_DATA. Update the SDO_GTYPE as follows:

```
UPDATE LRS_DATA a SET a.geometry.sdo_gtype = a.geometry.sdo_gtype + 400;
```

A.2 Ensuring That GeoRaster Works Properly After an Upgrade

If you are upgrading to Oracle Database 10g and if you use the GeoRaster feature of Oracle Spatial, Oracle XML DB Repository must be installed, and the value of the

COMPATIBILITY database initialization parameter must be 10.0 or greater. Before you use GeoRaster in the upgraded instance, ensure that both of these requirements are met, and if necessary reload the GeoRaster PL/SQL packages. Follow these steps:

1. Check to see if Oracle XML DB Repository is installed by checking if a schema exists in the database for a user named XDB. If the schema for user XDB exists, Oracle XML DB Repository is installed.
2. If Oracle XML DB Repository is not installed, install it with DBCA or by running the script `catqm.sql`, which is located in the `$ORACLE_HOME/rdbms/admin` directory.

For more information about installing and uninstalling Oracle XML DB Repository, see *Oracle XML DB Developer's Guide*.

3. Check the value of the Oracle initialization parameter `COMPATIBLE`.
4. If the value of the `COMPATIBLE` initialization parameter is less than 10.0, change the value to 10.0.

For more information about the `COMPATIBLE` initialization parameter, see *Oracle Database Reference*.

5. If you did not install Oracle XML DB Repository or change the value of the `COMPATIBLE` initialization parameter, stop; do not perform the remaining steps.

If you installed Oracle XML DB Repository or changed the value of the `COMPATIBLE` initialization parameter, or did both, go to the next step.

6. Go to the `$ORACLE_HOME/md/admin` directory.
7. Connect to the database as `SYS AS SYSDBA`.
8. Enter the following SQL statements:

```
ALTER SESSION SET CURRENT_SCHEMA=MDSYS;  
@prvtgrs.plb  
@sdogrxml.sql
```

Oracle Locator

Oracle Locator (also referred to as Locator) is a feature of Oracle Database 10g Standard Edition. Locator provides core features and services available in Oracle Spatial. It provides significant capabilities typically required to support Internet and wireless service-based applications and partner-based GIS solutions. Locator is not designed to be a solution for geographic information system (GIS) applications requiring complex spatial data management. If you need capabilities such as linear referencing, spatial functions, or coordinate system transformations, use Oracle Spatial instead of Locator.

Like Spatial, Locator is not designed to be an end-user application, but is a set of spatial capabilities for application developers.

Locator is available with both the Standard and Enterprise Editions of Oracle Database 10g. Spatial is a priced option available only with Oracle Database 10g Enterprise Edition. Spatial includes all Locator features as well as other features that are not available with Locator.

The installation of Locator depends on the successful and proper installation of Oracle *interMedia*. *interMedia* is installed and configured with Oracle Database 10g, although you can install *interMedia* manually if necessary, as documented in *Oracle interMedia User's Guide*. During the installation of *interMedia*, Locator is installed.

In general, Locator includes the data types, operators, and indexing capabilities of Oracle Spatial, along with a limited set of the functions and procedures of Spatial. The Locator features include the following:

- An object type (SDO_GEOMETRY) that describes and supports any type of geometry
- A spatial indexing capability that lets you create spatial indexes on geometry data
- Spatial operators (described in [Chapter 11](#)) that use the spatial index for performing spatial queries
- Some geometry functions and the [SDO_AGGR_MBR](#) spatial aggregate function
- Coordinate system support for explicit geometry and layer transformations ([SDO_CS.TRANSFORM](#) function and [SDO_CS.TRANSFORM_LAYER](#) procedure, described in [Chapter 13](#))
- Tuning functions and procedures (SDO_TUNE package, described in [Chapter 19](#))
- Spatial utility functions (SDO_UTIL package, described in [Chapter 20](#))
- Integration with Oracle Application Server 10g

For information about spatial concepts, the SDO_GEOMETRY object type, and indexing and loading spatial data, see Chapters 1 through 4 in this guide. For reference

and usage information about features supported by Locator, see the chapter or section listed in [Table B-1](#).

Table B-1 Spatial Features Supported for Locator

Spatial Feature	Described in
Function-based spatial indexing	Section 9.2
Table partitioning support for spatial indexes (including splitting, merging, and exchanging partitions and their indexes)	Section 4.1.4 and Section 4.1.5
Geodetic data support	Section 6.2 and Section 6.7
SQL statements for creating, altering, and deleting indexes	Chapter 10
Parallel spatial index builds (PARALLEL keyword with ALTER INDEX REBUILD and CREATE INDEX statements) (new with release 9.2)	Chapter 10
SDO_GEOMETRY object type methods	Section 2.3
Spatial operators (including SDO_JOIN, which is technically a table function but is documented with the operators)	Chapter 11
Implicit coordinate system transformations for operator calls where a window needs to be converted to the coordinate system of the queried layer.	Chapter 11
SDO_AGGR_MBR spatial aggregate function	Chapter 12
Coordinate system support for explicit geometry and layer transformations (SDO_CS.TRANSFORM function and SDO_CS.TRANSFORM_LAYER procedure)	Chapter 13
The following SDO_GEOM package functions and procedures: SDO_GEOM.SDO_DISTANCE, SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT, SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT	Chapter 15
Package (SDO_MIGRATE) to upgrade data from previous Spatial releases to the current release	Chapter 17
Tuning functions and procedures (SDO_TUNE package)	Chapter 19
Spatial utility functions (SDO_UTIL package)	Chapter 20
Object replication	<i>Oracle Database Advanced Replication</i>
Graphical tool for tuning spatial quadtree indexes (Spatial Index Advisor integrated application in Oracle Enterprise Manager)	Online help for Oracle Enterprise Manager

[Table B-2](#) lists Spatial features that are *not* supported for Locator, with the chapter in this guide or the separate manual that describes the feature.

Table B-2 Spatial Features Not Supported for Locator

Spatial Feature	Described in
SDO_GEOM package functions and procedures, except for those listed in Table B-1	Chapter 15
Spatial aggregate functions, except for any listed in Table B-1	Chapter 12
Linear referencing system (LRS) support	Chapter 7 (concepts and usage) and Chapter 16 (reference)

Table B–2 (Cont.) Spatial Features Not Supported for Locator

Spatial Feature	Described in
Spatial analysis and mining functions and procedures (SDO_SAM package)	Chapter 18
Geocoding support (SDO_GCDR package)	Chapter 5 (concepts and usage) and Chapter 14 (reference)
GeoRaster support	<i>Oracle Spatial GeoRaster</i>
Topology data model	<i>Oracle Spatial Topology and Network Data Models</i>
Network data model	<i>Oracle Spatial Topology and Network Data Models</i>

Although Locator is available on both the Standard and Enterprise Editions of Oracle Database 10g, some Locator features requires database features that are not available or are limited on the Standard Edition. Some of those Locator features and their availability are listed in [Table B–3](#).

Table B–3 Feature Availability with Standard and Enterprise Editions

Feature	Standard/Enterprise Edition Availability
Parallel spatial index builds	Supported with Enterprise Edition only.
Multimaster replication of SDO_GEOMETRY objects	Supported with Enterprise Edition only. (Single master/materialized view replication for SDO_GEOMETRY objects is supported with both Standard Edition and Enterprise Edition. See <i>Oracle Database Advanced Replication</i> for more information.)
Partitioned spatial indexes	Requires the Partitioning Option with Enterprise Edition. Not supported with Standard Edition.

Routing Engine

The Spatial routing engine enables you to host an XML-based Web service that provides the following features:

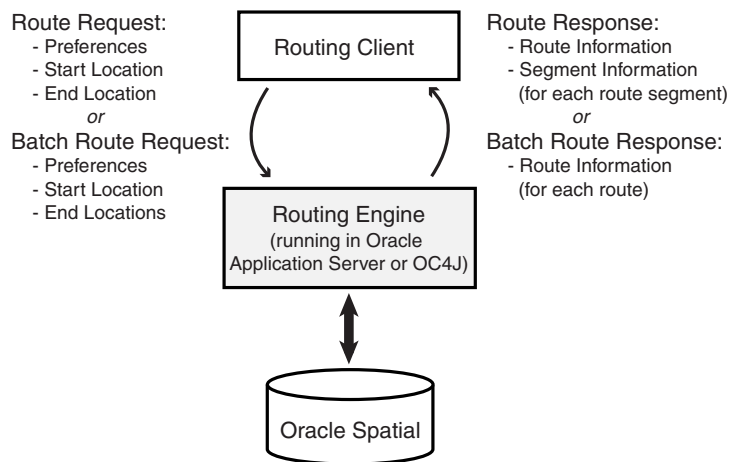
- For an individual route request (a start location and an end location): route information (driving distances, estimated driving times, and directions) between the two locations
- For a batch route request (multiple routes, with the same start location but different end locations): route information (driving distance and estimated driving time) for each route

For any request, the start and end locations are identified by addresses, geocoded results, or longitude/latitude coordinates.

The routing engine is implemented as a Java 2 Enterprise Edition (J2EE) Web application that you can deploy in either an Oracle Application Server or standalone Oracle Application Server Containers for J2EE (OC4J) environment.

Figure C-1 shows the basic flow of action with the routing engine: a client locates a remote routing engine instance, sends a route request, and processes the route response returned by the routing engine instance.

Figure C-1 Basic Flow of Action with the Spatial Routing Engine



This chapter contains the following major sections:

- [Section C.1, "Deploying and Configuring the Routing Engine"](#)
- [Section C.2, "Routing Engine XML API"](#)
- [Section C.3, "Data Structures Used by the Routing Engine"](#)

C.1 Deploying and Configuring the Routing Engine

To enable the routine engine to process routing requests and to generate responses, you must deploy the `routesserver.ear` file using OC4J or the Oracle Application Server. This section describes the basic steps.

1. Add the following element inside the `<web-site>` element in your `http-web-site.xml` or `default-web-site.xml` file of OC4J:

```
<web-app application="routesserver"
  name="web"
  load-on-startup="true"
  root="/routesserver"
  max-inactivity-time="no shutdown"
  shared="false" />
```

2. Use the Oracle Application Server console to deploy the `routesserver.ear` file, or add the following element inside the `<application-server>` element in the `server.xml` file of OC4J (replace `<ROUTE_SERVER_HOME>` accordingly):

```
<application name="routesserver"
  path="<ROUTE_SERVER_HOME>/routesserver.ear"
  auto-start="true" />
```

3. Add the following element inside the `<application-server>` element in the `server.xml` file of OC4J:

```
<max-http-connections value="10" />
```

It is important to limit the number of concurrent requests that the Oracle Route Server can process at any given time to prevent `java.lang.OutOfMemoryError` errors.

4. Start OC4J using the following command options:

```
-server
-Xms<HEAP_SIZE>
-Xmx<HEAP_SIZE>
-XX:NewSize=<YOUNG_GENERATION_SIZE>
-XX:MaxNewSize=<YOUNG_GENERATION_SIZE>
-Dsun.rmi.dgc.server.gcInterval=3600000
-Dsun.rmi.dgc.client.gcInterval=3600000
-verbose:gc (optional)
```

`<HEAP_SIZE>` must be at least 512 MB, and has a recommended size of at least 1024 MB (1 GB). Make sure that this memory is physical memory and not virtual memory.

`<YOUNG_GENERATION_SIZE>` should be one-fourth (25%) of the `<HEAP_SIZE>` value.

`-verbose:gc` will print all minor and major Java garbage collections. Monitoring these statistics could be useful for memory resource planning. If you find that garbage collections are occurring frequently or are lasting several seconds, you probably need to allocate more physical memory to the Java VM.

Note: The amount of memory the Java VM will need depends mostly on two parameters: the `<max-http-connections value="..." />` element in the `<application-server>` element in `server.xml`, and the `partition_cache_size_limit` parameter in `web.xml`.

The following command is an example that starts OC4J. Note that the `-config` flag is an OC4J command line parameter, not a VM option.

```
c:\jdk1.4.2\bin\java -server
                    -Xms1024m
                    -Xmx1024m
                    -XX:NewSize=256m
                    -XX:MaxNewSize=256m
                    -Dsun.rmi.dgc.server.gcInterval=3600000
                    -Dsun.rmi.dgc.client.gcInterval=3600000
                    -verbose:gc
                    -jar c:\oc4j\j2ee\home\oc4j.jar
                    -config c:\oc4j\j2ee\home\config\server.xml
```

5. Verify your deployment by visiting the URL in the following format:

```
http://<hostname>:<port>/routeserver
```

You should see a welcome page. You should also see a message in the console window in which you started OC4J indicating that the Oracle Route Server was successfully initialized.

If you do not see a welcome message, the route server is probably not configured properly to run in your environment. In this case, edit the `<ROUTE_SERVER_HOME>/routeserver/web/WEB-INF/web.xml` file to reflect your environment and your preferences. (The `web.xml` file is inside the `routeserver.ear` file, and it will not be visible until OC4J expands it into the route server directory structure under `<ROUTE_SERVER_HOME>`.) When you are finished editing, restart OC4J and verify your deployment.

6. Consult the supplied examples. The page `http://<hostname>:<port>/routeserver/` has links at the bottom in a section named Test Samples. These examples demonstrate various capabilities of the Oracle Route Server. This is the best way to learn the XML API, which is described in [Section C.2](#).

C.2 Routing Engine XML API

This section explains how to submit route requests in XML format to the routing engine, and it describes the XML document type definitions (DTDs) for the route requests (input) and responses (output). XML is widely used for transmitting structured documents using the HTTP protocol. If an HTTP request (GET or POST method) is used, it is assumed the request has a parameter named `xml_request` whose value is a string containing the XML document for the request.

A request to the routing engine servlet has the following format:

```
http://hostname:port/route-server-servlet-path?xml_request=xml-request
```

In this format:

- *hostname* is the network path of the server on which the routing engine is running.

- *port* is the port on which the application server listens.
- *route-server-servlet-path* is the routing engine servlet path (for example, `routeserver/servlet/RouteServerServlet`).
- *xml-request* is the URL-encoded XML request submitted using the HTML GET or POST method.

The input XML is required for all requests. The output will be an XML document.

In an input *route* (as opposed to batch route) request, you must specify a route ID, and you can specify one or more of the following attributes:

- *route_preference*: `fastest` or `shortest` (default). (Note that for *batch* route requests, the default is `fastest`.)
- *road_preference*: `highway` (default) or `local`
- *return_hierarchical_directions* (whether to return hierarchical directions): `true` or `false` (default)
- *return_driving_directions* (whether to return driving directions): `true` (default) or `false`
- *return_route_geometry* (whether to return the line string coordinates for the route): `true` or `false` (default)
- *return_detailed_geometry*: `true` (default; returns detailed geometries) or `false` (returns generalized geometries)
- *distance_unit*: `kilometer`, `mile` (default), or `meter`
- *time_unit*: `hour`, `minute` (default), or `second`
- *pre_geocoded_locations* (whether the start and end locations are input locations (address specifications or points) or previously geocoded locations): `true` (previously geocoded locations) or `false` (default; input locations)

In an input *batch route* request, you must specify a request ID, a start location, and one or more end locations. Each location must have an ID attribute. You can also specify one or more of the following attributes for the batch route request:

- *route_preference*: `fastest` (default) or `shortest`. (Note that for *individual* route requests, the default is `shortest`.)
- *road_preference*: `highway` (default) or `local`
- *distance_unit*: `kilometer`, `mile` (default), or `meter`
- *time_unit*: `hour`, `minute` (default), or `second`
- *sort_by_distance* (whether to sort the returned routes in ascending order by distance of the end location from the start location): `true` or `false` (default)
- *cutoff_distance* (returning only routes where the end location is less than or equal to a specified number of distance units from the start location): (number; default = no limit)
- *pre_geocoded_locations* (whether the start and end locations are input locations (address specifications or points) or previously geocoded locations): `true` (previously geocoded locations) or `false` (default; input locations)

This section contains the following subsections:

- [Section C.2.1, "Route Request and Response Examples"](#)
- [Section C.2.2, "Route Request DTD"](#)

- [Section C.2.3, "Route Response DTD"](#)
- [Section C.2.4, "Batch Route Request and Response Examples"](#)
- [Section C.2.5, "Batch Route Request DTD"](#)
- [Section C.2.6, "Batch Route Response DTD"](#)
- [Section C.2.7, "Geocoding Request and Response DTDs"](#)

C.2.1 Route Request and Response Examples

This section contains XML examples of route requests and the responses generated by those requests. One request uses specified addresses, another uses points specified by longitude and latitude coordinates, and another uses previously geocoded locations. For reference information about the available elements and attributes, see [Section C.2.2](#) for requests and [Section C.2.3](#) for responses.

[Example C-1](#) shows a request for the fastest route, preferably using highways, between two offices at specified addresses (in Waltham, Massachusetts and Nashua, New Hampshire), with driving directions for each segment, and using miles for distances and minutes for times.

Example C-1 Route Request with Specified Addresses

```
<?xml version="1.0" standalone="yes"?>
<route_request
  id="8"
  route_preference="fastest"
  road_preference="highway"
  return_driving_directions="true"
  distance_unit="mile"
  time_unit="minute">
  <start_location>
    <input_location id="1">
      <input_address>
        <us_form1
          street="1000 Winter St"
          lastline="Waltham, MA" />
        </input_address>
      </input_location></start_location>
    <end_location>
      <input_location id="2">
        <input_address>
          <us_form1
            street="1 Oracle Dr"
            lastline="Nashua, NH" />
          </input_address>
        </input_location>
      </end_location>
    </route_request>
```

[Example C-2](#) shows the response generated by the request in [Example C-1](#). (The output is reformatted for readability.)

Example C-2 Route Response with Specified Addresses

```
<?xml version="1.0" ?>
<route_response>
  <route
    id="8"
```

```
        step_count="14"
        distance="30.28667355371901"
        distance_unit="mile"
        time="35.02037760416667"
        time_unit="minute">
<segment
  sequence="1"
  instruction="Start out on WINTER ST (Going South) "
  distance="1.2041612436793172"/>
<segment
  sequence="2"
  instruction="Stay STRAIGHT to go onto TOTTEN POND RD (Going East) "
  distance="0.08879983757738225"/>
<segment
  sequence="3"
  instruction="Turn LEFT onto WYMAN ST (Going North) "
  distance="0.24681569656886923"/>
<segment
  sequence="4"
  instruction="Take I-95 N RAMP toward PEABODY"
  distance="0.23440010735937208"/>
<segment
  sequence="5"
  instruction="Merge onto I-95/RT-128 (Going North) "
  distance="6.002288440990454"/>
<segment
  sequence="6"
  instruction="Continue on I-95/RT-128"
  distance="0.0"/>
<segment
  sequence="7"
  instruction="Stay STRAIGHT to go onto 32B/32A (Going East) "
  distance="0.15052764594854906"/>
<segment
  sequence="8"
  instruction="Take EXIT 32A toward LOWELL"
  distance="0.032767910543403965"/>
<segment
  sequence="9"
  instruction="Stay STRAIGHT to go onto RAMP (Going East) "
  distance="0.27877937515534706"/>
<segment
  sequence="10"
  instruction="Turn LEFT onto US-3 (Going Northwest) "
  distance="20.66104112133381"/>
<segment
  sequence="11"
  instruction="Stay STRAIGHT to go onto FREDERICK E EVERETT
              TPKE/US-3 (Going Northwest) "
  distance="0.00588619663828994"/>
<segment
  sequence="12"
  instruction="Take EXIT 1 toward SO NASHUA"
  distance="0.5504892461007892"/>
<segment
  sequence="13"
  instruction="Turn LEFT onto SPIT BROOK RD (Going West) "
  distance="0.5032054891878457"/>
<segment
  sequence="14"
```

```

        instruction="Turn RIGHT onto ORACLE DR (Going North)"
        distance="0.3275097635011146"/>
    </route>
</route_response>

```

[Example C-3](#) shows a request for the fastest route, preferably using highways, between two locations specified as longitude/latitude points, with driving directions for each segment, and using meters for distances and seconds for times. (The points are associated with two locations in San Francisco, California: the World Trade Center and 100 Flower Street.)

Example C-3 Route Request with Specified Longitude/Latitude Points

```

<?xml version="1.0" standalone="yes"?>
<route_request id="8"
    route_preference="shortest"
    road_preference="highway"
    return_driving_directions="true"
    distance_unit="meter"
    time_unit="second"
    return_route_geometry="true"
>
  <start_location>
    <input_location id="1" longitude="-122.39382" latitude="37.79518" />
  </start_location>
  <end_location>
    <input_location id="2" longitude="-122.4054826" latitude="37.7423566" />
  </end_location>
</route_request>

```

[Example C-4](#) shows the response generated by the request in [Example C-3](#). (The output is reformatted for readability.)

Example C-4 Route Response with Specified Longitude/Latitude Points

```

<route_response>
  <route id="8" step_count="13" distance="7261.4423828125" distance_unit="meter"
    time="441.9170837402344" time_unit="second">
    <route_geometry>
      <LineString>
        <coordinates>
-122.39381999996483,37.79517999996185 -122.39382,37.79518 -122.39458,37.79598 -122.39469,37.796
-122.39474,37.796 -122.39479,37.79599 -122.39483,37.79591 -122.39483,37.79579 -122.39462,37.79539
-122.39424,37.79488 -122.39338,37.79434 -122.39311,37.79413 -122.39275,37.79384 -122.39258,37.79368
-122.39171,37.79297 -122.39145,37.79273 -122.39127,37.79248 -122.3912,37.79235 -122.39107,37.79208
-122.39098,37.79185 -122.39088,37.79161 -122.39075,37.79138 -122.39048,37.79105 -122.3901,37.79079
-122.38918,37.79001 -122.38877,37.78968 -122.38857,37.78948 -122.38939,37.78882 -122.39024,37.78815
-122.39113,37.78745 -122.39192,37.7868 -122.39284,37.78606 -122.39372,37.78535 -122.39406,37.78507
-122.39511,37.78426 -122.39565,37.78383 -122.39621,37.78337 -122.39728,37.78252 -122.39824,37.78177
-122.39955,37.78075 -122.39963,37.78032 -122.3997,37.78011 -122.39984,37.77991 -122.40071,37.77899
-122.40085,37.77888 -122.40129,37.77855 -122.40182,37.77815 -122.40245,37.77776 -122.40302,37.77737
-122.40375,37.77695 -122.40433,37.77657 -122.40529,37.77592 -122.40581,37.7755 -122.40605,37.77524
-122.4063,37.77493 -122.40656,37.7744 -122.40671,37.7739 -122.40683,37.77312 -122.40671,37.77264
-122.4066,37.77216 -122.40634,37.77151 -122.40594,37.77074 -122.40573,37.77022 -122.4055,37.76958
-122.40547,37.76913 -122.40541,37.76843 -122.40542,37.76791 -122.40547,37.76743 -122.40541,37.76715
-122.40526,37.76579 -122.4051,37.7645 -122.40513,37.76404 -122.40519,37.76356 -122.40544,37.7629
-122.40561,37.76257 -122.40586,37.76218 -122.40619,37.76161 -122.40636,37.7612 -122.40648,37.76063
-122.40642,37.75996 -122.40633,37.75965 -122.4061,37.75918 -122.40574,37.75875 -122.40543,37.75846
-122.4045,37.75778 -122.40402,37.75735 -122.4038,37.75712 -122.40365,37.75688 -122.40344,37.75645
-122.4033,37.75588 -122.40326,37.75537 -122.40316,37.75437 -122.40304,37.75256 -122.40376,37.7502

```

```

-122.40384,37.74976 -122.40396,37.74969 -122.40454,37.74947 -122.40468,37.74933 -122.40474,37.74921
-122.40471,37.74902 -122.4045,37.74873 -122.40417,37.74839 -122.404,37.7482 -122.40378,37.74799
-122.40376,37.74781 -122.40428,37.74623 -122.40428,37.74598 -122.40417,37.74557 -122.40419,37.74483
-122.40431,37.74423 -122.40443,37.74396 -122.40468,37.74353 -122.40509,37.74294 -122.40472,37.74274
-122.40512,37.7422 -122.40548260000706,37.74235680000305
  </coordinates>
</LineString>
</route_geometry>
<segment sequence="1" instruction="Start out on THE EMBARCADERO (Going
  Northwest)" distance="5.246016371529549E-6"/>
<segment sequence="2" instruction="Stay STRAIGHT to go onto THE
  EMBARCADERO/WORLD TRADE CTR/FERRY PLZ/FERRY
  BLDG (Going Northwest)" distance="111.19815063476562"/>
<segment sequence="3" instruction="Turn LEFT onto RAMP (Going Southwest)"
  distance="41.756561279296875"/>
<segment sequence="4" instruction="Turn LEFT onto THE EMBARCADERO (Going
  Southeast)" distance="905.924072265625"/>
<segment sequence="5" instruction="Turn RIGHT onto HARRISON ST (Going
  Southwest)" distance="1369.1490478515625"/>
<segment sequence="6" instruction="Take I-80 W RAMP toward SAN JOSE"
  distance="225.425048828125"/>
<segment sequence="7" instruction="Turn SLIGHT RIGHT onto I-80/JAMES LICK
  SKWY (Going Southwest)" distance="1528.181396484375"/>
<segment sequence="8" instruction="Stay STRAIGHT to go onto
  US-101/JAMES LICK FWY (Going South)" distance="1765.10498046875"/>
<segment sequence="9" instruction="Turn SLIGHT RIGHT onto RAMP (Going
  South)" distance="481.18505859375"/>
<segment sequence="10" instruction="Turn LEFT onto BAY SHORE BLVD (Going
  Southeast)" distance="688.142578125"/>
<segment sequence="11" instruction="Turn LEFT onto OAKDALE AVE (Going
  Southeast)" distance="39.44921875"/>
<segment sequence="12" instruction="Turn RIGHT onto PATTERSON ST (Going
  Southwest)" distance="69.53564453125"/>
<segment sequence="13" instruction="Turn RIGHT onto FLOWER ST (Going
  Northwest)" distance="36.39051818847656"/>
</route>
</route_response>

```

Example C-5 shows a request for the route, with driving directions, where the start and end locations are previously geocoded locations that are about one-half mile apart in Boston, Massachusetts.

Example C-5 Route Request with Previously Geocoded Locations

```

<?xml version="1.0" standalone="yes"?>
<route_request id="8"
  route_preference="shortest"
  road_preference="highway"
  return_driving_directions="true"
  distance_unit="mile"
  time_unit="minute"
  pre_geocoded_locations="true">
  <start_location>
    <pre_geocoded_location id="1">
      <edge_id>22161661</edge_id>
      <percent>.5</percent>
      <side>L</side>
    </pre_geocoded_location>
  </start_location>
</end_location>

```



```

    <pre_geocoded_location id="2">
      <edge_id>22104391</edge_id>
      <percent>.5</percent>
      <side>R</side>
    </pre_geocoded_location>
  </end_location>
</route_request>

```

[Example C-6](#) shows the response to the request in [Example C-5](#). (The output is reformatted for readability.)

Example C-6 Route Response with Previously Geocoded Locations

```

<?xml version="1.0" ?>
<route_response>
  <route
    id="8"
    step_count="5"
    distance="0.5848966065287509"
    distance_unit="mile"
    time="1.1866167704264323"
    time_unit="minute">
    <segment
      sequence="1"
      instruction="Start out on HUNTINGTON AVE (Going Southeast)"
      distance="0.005477076104790563" />
    <segment
      sequence="2"
      instruction="Turn LEFT onto AVENUE OF THE ARTS/HUNTINGTON
        AVE/RT-9 (Going Northeast)"
      distance="0.006677015642704102" />
    <segment
      sequence="3"
      instruction="Turn RIGHT onto PUBLIC ALLEY 405 (Going Southeast)"
      distance="0.05267257088346108" />
    <segment
      sequence="4"
      instruction="Turn RIGHT onto ST BOTOLPH ST (Going Southwest)"
      distance="0.010097520017923165" />
    <segment
      sequence="5"
      instruction="Turn RIGHT onto MASSACHUSETTS AVE (Going Northwest)"
      distance="0.5099724250650759" />
  </route>
</route_response>

```

C.2.2 Route Request DTD

The following is the complete DTD for a route request. The main elements and attributes of the DTD are explained in sections that follow.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- geocoder.dtd includes gmlfeature.dtd. These define the
ELEMENTS input_address and those in Feature, FeatureMember,
and FeatureCollection that are used in geoFeature,
geoFeatureCollection, and geoFeatureMember.
-->
<!ENTITY % GEOCODERDTD SYSTEM "geocoder.dtd">
%GEOCODERDTD;

```

```

<!--
    input_location element is defined in geocoder.dtd.
    GeometryClasses is defined in gmlgeometry.dtd.
-->
<!ELEMENT route_request (start_location, end_location)>
<!ATTLIST route_request
    vendor CDATA "Oracle"
    id CDATA #REQUIRED
    route_preference (FASTEST|SHORTEST) #IMPLIED
    road_preference (HIGHWAY|LOCAL) #IMPLIED
    return_driving_directions (TRUE|FALSE) #IMPLIED
    return_hierarchical_driving_directions (TRUE|FALSE) #IMPLIED
    return_route_geometry (TRUE|FALSE) #IMPLIED
    return_detailed_geometries (TRUE|FALSE) #IMPLIED
    return_segment_geometry (TRUE|FALSE) #IMPLIED
    language CDATA #IMPLIED
    distance_unit (KM|MILE|METER) #IMPLIED
    time_unit (HOUR|MINUTE|SECOND) #IMPLIED>

<!-- Following are alternatives for specifying the location. Use
input_location when you want to represent a location with a
street address. Use longitude_latitude_location when you want to
specify a location by longitude and latitude coordinates.
If you have already geocoded the location,
you can use information from the geocoder response to
construct a pre_geocoded_location element.
The geocoder returns:
- An edge_id (integer that is the road segment identifier)
- A side ('L' or 'R' - left or right side)
- A percent (floating-point number 0.0 to 1.0 representing
the fraction of the length from the start of the road
segment to this location.
-->
<!ELEMENT pre_geocoded_location (edge_id, percent, side)>
<!ATTLIST pre_geocoded_location id CDATA #REQUIRED>
<!ELEMENT longitude_latitude_location (longitude, latitude)>

<!ELEMENT start_location (input_location|pre_geocoded_location|longitude_latitude_
location)>
<!ELEMENT end_location (input_location|pre_geocoded_location|longitude_latitude_
location)>

```

C.2.2.1 route_request Element

The <route_request> element has the following definition:

```
<!ELEMENT route_request (start_location, end_location)>
```

The root element of a route request is always named route_request.

The <start_location> child element specifies the start location for the route, as an address specification, a geocoded address, or longitude/latitude coordinates.

The <end_location> child element specifies the end location for the route, as an address specification, a geocoded address, or longitude/latitude coordinates.

C.2.2.2 route_request Attributes

The root element <route_request> has a number of attributes, most of them optional. The attributes are defined as follows:

```
<!ATTLIST route_request
```

```

vendor CDATA "Oracle"
id CDATA #REQUIRED
route_preference (FASTEST|SHORTEST) #IMPLIED
road_preference (HIGHWAY|LOCAL) #IMPLIED
return_driving_directions (TRUE|FALSE) #IMPLIED
return_route_geometry (TRUE|FALSE) #IMPLIED
return_segment_geometry (TRUE|FALSE) #IMPLIED
language CDATA #IMPLIED
distance_unit (KM|MILE|METER) #IMPLIED
time_unit (HOUR|MINUTE|SECOND) #IMPLIED
pre_geocoded_locations (TRUE|FALSE) #IMPLIED
return_hierarchical_driving_directions (TRUE|FALSE) #IMPLIED
return_detailed_geometries (TRUE|FALSE) #IMPLIED>

```

`vendor` is an optional attribute whose default value identifies the routing provider as Oracle.

`id` is a required attribute that specifies an identification number to be associated with the request.

`route_preference` is an optional attribute that specifies whether you want the route with the lowest estimated driving time (FASTEST) or the route with the shortest driving distance (SHORTEST, the default).

`road_preference` is an optional attribute that specifies whether you want the route to use highways (HIGHWAY, the default) or local roads (LOCAL) when a choice is available.

`return_driving_directions` is an optional attribute that specifies whether you want driving directions for the route. TRUE (the default) returns driving directions; FALSE does not return driving directions.

`return_route_geometry` is an optional attribute that specifies whether you want the coordinates of the line string that represents the route. TRUE returns the coordinates; FALSE (the default) does not return the coordinates.

`return_segment_geometry` is currently ignored.

`language` is currently ignored.

`distance_unit` is an optional attribute that specifies the unit of measure for distance values that are returned: KM for kilometer, MILE (the default) for mile, or METER for meter.

`time_unit` is an optional attribute that specifies the unit for time values that are returned: HOUR for hour, MINUTE (the default) for minute, or SECOND for second.

`pre_geocoded_locations` is an optional attribute that indicates how the start and end locations are specified. TRUE means that both are previously geocoded locations specified using the `<pre_geocoded_location>` element; FALSE (the default) means that both are addresses specified using the `<input_address>` or `<longitude_latitude_location>` element.

`return_hierarchical_driving_directions` is an optional attribute that specifies whether to return the driving directions as an expandable and collapsible hierarchy or as a list with no hierarchy. TRUE means to return a hierarchy; FALSE (the default) means to return a list with no hierarchy.

`return_detailed_geometries` is an optional attribute that indicates the level of detail to be included in returned geometries. TRUE (the default) returns detailed geometries; FALSE returns generalized geometries (and usually smaller).

C.2.2.3 input_location Element

The `<input_location>` element specifies an address in a format that satisfies the Oracle Spatial geocoding request DTD, which is described in [Section C.2.7.1](#). You can specify the input location using either the `<Point>` element or the `<input_address>` element. [Example C-1](#) in [Section C.2.1](#) shows the start and end addresses specified using the `<input_location>` element and its child element `<input_address>`.

To use the `<input_location>` element, you must ensure that the value of the `pre_geocoded_locations` attribute is `FALSE` (the default) in the `<route_request>` element. To specify the start location and the end location, you can use the `<input_location>` element for both, the `<longitude_latitude_location>` element (described in [Section C.2.2.5](#)) for both, or the `<input_location>` element for one and the `<longitude_latitude_location>` element for the other.

C.2.2.4 pre_geocoded_location Element

The `<pre_geocoded_location>` element specifies a geocoded location in terms of how far along a street (an edge) the address is and on which side of the street.

[Example C-5](#) in [Section C.2.1](#) shows the start and end addresses specified using the `<pre_geocoded_location>` element.

To use the `<pre_geocoded_location>` element, you must specify `pre_geocoded_locations="TRUE"` in the `<route_request>` element, and you must use the `<pre_geocoded_location>` element to specify both the start and end locations.

C.2.2.5 longitude_latitude_location Element

The `<longitude_latitude_location>` element specifies longitude and latitude coordinates for the location.

To use the `<longitude_latitude_location>` element, you must ensure that the value of the `pre_geocoded_locations` attribute is `FALSE` (the default) in the `<route_request>` element. To specify the start location and the end location, you can use the `<input_location>` element (described in [Section C.2.2.3](#)) for both, the `<longitude_latitude_location>` element for both, or the `<input_location>` element for one and the `<longitude_latitude_location>` element for the other.

C.2.3 Route Response DTD

The following is the complete DTD for a route response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- route_response DTD includes the gmlgeometry DTD
      as an external entity reference.
-->
<!ENTITY % GMLGEOMETRYDTD SYSTEM "gmlgeometry.dtd">
  %GMLGEOMETRYDTD;

<!ELEMENT route_response (route | router_error)>

<!ELEMENT route (route_geometry?, segment+)>
<!-- ATTLLIST route id CDATA #REQUIRED
              step_count CDATA #IMPLIED
              time CDATA #IMPLIED
              distance CDATA #IMPLIED-->

<!ELEMENT router_error EMPTY>
<!-- ATTLLIST router_error
```

```

        id CDATA #REQUIRED
        error_code CDATA #IMPLIED
        error_msg CDATA #IMPLIED>

<!ELEMENT route_geometry (LineString | MultiLineString)?>

<!ELEMENT segment segment*, (LineString | MultiLineString)?>
<!ATTLIST segment sequence CDATA #REQUIRED
                instruction CDATA #IMPLIED
                distance CDATA #IMPLIED>

```

C.2.4 Batch Route Request and Response Examples

This section contains XML examples of batch route requests and the responses generated by those requests. One request uses specified addresses, and the other request uses previously geocoded locations. For reference information about the available elements and attributes, see [Section C.2.5](#) for requests and [Section C.2.6](#) for responses.

[Example C-7](#) shows a batch route request using specified addresses. The request is for the fastest routes, preferably using highways, between an office in Waltham, Massachusetts and three end locations (an Oracle office in Nashua, New Hampshire; the town offices in Concord, Massachusetts; and Boston City Hall), using miles for distances and minutes for times. The request calls for the returned routes to be sorted by distance between the start and end location, and for no routes over 35 miles to be returned.

Example C-7 Batch Route Request with Specified Addresses

```

<?xml version="1.0" standalone="yes"?>
<batch_route_request
  id="8"
  route_preference="fastest"
  road_preference="highway"
  return_driving_directions="false"
  sort_by_distance = "true"
  cutoff_distance="35"
  distance_unit="mile"
  time_unit="minute">
  <start_location>
    <input_location
      id="1">
      <input_address>
        <us_form1
          street="1000 Winter St"
          lastline="Waltham, MA" />
        </input_address>
      </input_location>
    </start_location>
    <end_location>
      <input_location id="10">
        <input_address>
          <us_form1
            street="1 Oracle Dr"
            lastline="Nashua, NH" />
          </input_address>
        </input_location>
      </end_location>
    <end_location>
      <input_location

```

```

        id="11">
        <input_address>
        <us_form1
        street="22 Monument Sq"
        lastline="Concord, MA" />
        </input_address>
    </input_location>
</end_location>
<end_location>
    <input_location
        id="12">
        <input_address>
        <us_form1
        street="1 City Hall Plaza"
        lastline="Boston, MA" />
        </input_address>
    </input_location>
</end_location>
</batch_route_request>

```

[Example C-8](#) shows the response generated by the request in [Example C-7](#). (The output is reformatted for readability.)

Example C-8 Batch Route Response with Specified Addresses

```

<?xml version="1.0" standalone="yes" ?>
<batch_route_response
    id="8">
    <route
        id="11"
        step_count="0"
        distance="9.132561517429938"
        distance_unit="mile"
        time="12.4705078125"
        time_unit="minute" />
    <route
        id="12"
        step_count="0"
        distance="17.74747391140558"
        distance_unit="mile"
        time="20.413236490885417"
        time_unit="minute" />
    <route
        id="10"
        step_count="0"
        distance="30.28667355371901"
        distance_unit="mile"
        time="35.02037760416667"
        time_unit="minute" />
</batch_route_response>

```

[Example C-9](#) shows a batch route request using previously geocoded locations. The request is for the shortest routes, preferably using highways, between one location and three other locations, using miles for distances and minutes for times. The request calls for the returned routes to be sorted by distance between the start and end location, and for no routes over 50 miles to be returned.

Example C-9 Batch Route Request with Previously Geocoded Locations

```

<?xml version="1.0" standalone="yes"?>

```

```

<batch_route_request id="8"
    route_preference="shortest"
    road_preference="highway"
    return_driving_directions="false"
    distance_unit="mile"
    time_unit="minute"
    pre_geocoded_locations="true"
    cutoff_distance="50"
    sort_by_distance="true">
  <start_location>
    <pre_geocoded_location id="1">
      <edge_id>22161661</edge_id>
      <percent>.5</percent>
      <side>L</side>
    </pre_geocoded_location>
  </start_location>
  <end_location>
    <pre_geocoded_location id="2">
      <edge_id>22104391</edge_id>
      <percent>.5</percent>
      <side>R</side>
    </pre_geocoded_location>
  </end_location>
  <end_location>
    <pre_geocoded_location id="3">
      <edge_id>22160808</edge_id>
      <percent>.5</percent>
      <side>L</side>
    </pre_geocoded_location>
  </end_location>
  <end_location>
    <pre_geocoded_location id="4">
      <edge_id>22325991</edge_id>
      <percent>.5</percent>
      <side>R</side>
    </pre_geocoded_location>
  </end_location>
</batch_route_request>

```

Example C-10 shows the response to the request in [Example C-9](#). Only two routes are returned, because the third route is longer than the specified cutoff distance of 50 miles. (The output is reformatted for readability.)

Example C-10 Batch Route Response with Previously Geocoded Locations

```

<?xml version="1.0" standalone="yes" ?>
<batch_route_response id="8">
  <route
    id="2"
    step_count="0"
    distance="0.5848966065287509"
    distance_unit="mile"
    time="1.1866167704264323"
    time_unit="minute" />
  <route
    id="4"
    step_count="0"
    distance="41.09054596719071"
    distance_unit="mile"
    time="45.4477294921875"

```

```

        time_unit="minute" />
</batch_route_response>

```

C.2.5 Batch Route Request DTD

The following is the complete DTD for a batch route request. The main elements and attributes of the DTD are explained in sections that follow.

```

<!ENTITY % GEOCODERDTD SYSTEM "geocoder.dtd">
%GEOCODERDTD;
<!-- input_location element is defined in geocoder.dtd -->

<!ELEMENT batch_route_request (start_location, end_location+)>
<!ATTLIST batch_route_request
    vendor CDATA "Oracle"
    id CDATA #REQUIRED
    route_preference (FASTEST | SHORTEST ) #IMPLIED
    road_preference (HIGHWAY | LOCAL) #IMPLIED
    distance_unit (KM | MILE | METER ) #IMPLIED
    time_unit (HOUR | MINUTE | SECOND) #IMPLIED
    sort_by_distance (TRUE | FALSE) #IMPLIED
    cutoff_distance CDATA #IMPLIED>

<!-- Following are alternatives for specifying the location. Use
input_location when you want to represent a location with a
street address. Use longitude_latitude_location when you want to
specify a location by longitude and latitude coordinates.
If you have already geocoded the location,
you can use information from the geocoder response to
construct a pre_geocoded_location element.
The geocoder returns:
- an edge_id (integer that is the road segment identifier)
- a side ('L' or 'R' - left or right side)
- a percent (floating-point number 0.0 to 1.0 representing
the fraction of the length from the start of the road
segment to this location.
-->
<!ELEMENT pre_geocoded_location (edge_id, percent, side)>
<!ATTLIST pre_geocoded_location id CDATA #REQUIRED>
<!ELEMENT longitude_latitude_location (longitude, latitude)>

<!ELEMENT start_location (input_location|pre_geocoded_location|longitude_latitude_
location)>
<!ELEMENT end_location (input_location|pre_geocoded_location|longitude_latitude_
location)>
<!-- IMPORTANT VALIDITY CONSTRAINT: each of the input_location
elements that are children of end_location MUST contain
the id attribute. Normally, the id attribute is optional.
If an id is not present, an exception will result.
Also, each id must be unique within a batch_route_request.
Otherwise, the request will yield unpredictable results.
-->

```

C.2.5.1 batch_route_request Element

The <batch_route_request> element has the following definition:

```

<!ELEMENT batch_route_request (start_location, end_location+)>

```

The root element of a route request is always named batch_route_request.

The `<start_location>` child element specifies the start location for the route, as an address specification, a geocoded address, or longitude/latitude coordinates.

Each of the one or more `<end_location>` child elements specifies the end location for the route, as an address specification, a geocoded address, or longitude/latitude coordinates.

C.2.5.2 batch_route_request Attributes

The root element `<batch_route_request>` has a number of attributes, most of them optional. The attributes are defined as follows:

```
<!ATTLIST batch_route_request
  vendor CDATA "Oracle"
  id CDATA #REQUIRED
  route_preference (FASTEST|SHORTEST) #IMPLIED
  road_preference (HIGHWAY|LOCAL) #IMPLIED
  distance_unit (KM|MILE|METER) #IMPLIED
  time_unit (HOURL|MINUTE|SECOND) #IMPLIED
  sort_by_distance (TRUE | FALSE) #IMPLIED
  cutoff_distance CDATA #IMPLIED>
pre_geocoded_locations (TRUE|FALSE) #IMPLIED>
```

Most `<batch_route_request>` attributes have the same meaning as their counterpart `<route_request>` attributes, which are explained in [Section C.2.5.2](#). In addition, the `sort_by_distance` and `cutoff_distance` attributes do not apply to single route requests.

`sort_by_distance` is an optional attribute that specifies whether you want the routes returned in ascending order by distance of the end location from the start location. `TRUE` sorts the returned routes by distance; `FALSE` (the default) does not sort the returned routes by distance.

`cutoff_distance` is an optional attribute that causes routes to be returned only where the end location is less than or equal to a specified distance from the start location. By default, all routes are returned.

Note: If a route is within the specified `cutoff_distance` value but would generate a `<router_error>` element in the response (see [Section C.2.6](#)), the route is removed from the response and not shown.

C.2.6 Batch Route Response DTD

The following is the complete DTD for a batch route response:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT batch_route_response (route | route_error)+ >
<!ATTLIST batch_route_response id CDATA #REQUIRED>
<!ELEMENT route EMPTY>
<!ATTLIST route
  id CDATA #REQUIRED
  step_count CDATA #IMPLIED
  distance CDATA #IMPLIED
  distance_unit CDATA #IMPLIED
  time CDATA #IMPLIED
  time_unit CDATA #IMPLIED>
<!ELEMENT router_error EMPTY>
<!ATTLIST router_error
  id CDATA #REQUIRED
  error_code CDATA #IMPLIED
```

```
error_msg CDATA #IMPLIED>
```

C.2.7 Geocoding Request and Response DTDs

This section presents the DTDs for requests to geocode an address and for responses to these requests. These DTDs are supported only for use with route requests, and they are not explained in detail.

C.2.7.1 Geocoding Request DTD

The DTD for a request to geocode an address is as follows:

```
<!-- geocode_request DTD includes the GML Feature
DTD as an external entity reference. The complete
URL for the DTD is:
http://www.opengis.org/techno/specs/00-029/gmlfeature.dtd
-->
<!ENTITY % GMLFEATUREDTD SYSTEM "gmlfeature.dtd">
%GMLFEATUREDTD;
<!ELEMENT geocode_request (address_list)>
<!ATTLIST geocode_request >
<!ELEMENT address_list (input_location+)>
<!ELEMENT input_location (Point | input_address)>
<!ATTLIST input_location
  id CDATA #IMPLIED
  multimatch_number CDATA "4"
>

<!ELEMENT Point EMPTY>
<!ATTLIST Point
  longitude CDATA #IMPLIED
  latitude CDATA #IMPLIED
>

<!ELEMENT input_address (us_form1 | us_form2 | gdf_form | gen_form | unformatted)>
<!ATTLIST input_address match_mode CDATA #IMPLIED >
<!ELEMENT gdf_form EMPTY>
<!ATTLIST gdf_form
  name CDATA #IMPLIED
  street CDATA #IMPLIED
  intersecting_street CDATA #IMPLIED
  builtup_area CDATA #IMPLIED
  order8_area CDATA #IMPLIED
  order2_area CDATA #IMPLIED
  order1_area CDATA #IMPLIED
  country CDATA #IMPLIED
  postal_code CDATA #IMPLIED
  postal_addon_code CDATA #IMPLIED
>

<!ELEMENT gen_form EMPTY>
<!ATTLIST gen_form
  name CDATA #IMPLIED
  street CDATA #IMPLIED
  intersecting_street CDATA #IMPLIED
  sub_area CDATA #IMPLIED
  city CDATA #IMPLIED
  region CDATA #IMPLIED
  country CDATA #IMPLIED
  postal_code CDATA #IMPLIED
```

```

    postal_addon_code CDATA #IMPLIED
  >

  <!ELEMENT us_form1 EMPTY>
  <!ATTLIST us_form1
    name CDATA #IMPLIED
    street CDATA #IMPLIED
    intersecting_street CDATA #IMPLIED
    lastline CDATA #IMPLIED
  >

  <!ELEMENT us_form2 EMPTY>
  <!ATTLIST us_form2
    name CDATA #IMPLIED
    street CDATA #IMPLIED
    intersecting_street CDATA #IMPLIED
    city CDATA #IMPLIED
    state CDATA #IMPLIED
    zip_code CDATA #IMPLIED
  >

  <!ELEMENT unformatted (address_line) >
  <!ATTLIST unformatted country CDATA #IMPLIED >
  <!ELEMENT address_line EMPTY >
  <!ATTLIST value #REQUIRED >

```

C.2.7.2 Geocoding Response DTD

The DTD for a response generated by a request to geocode an address is as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT geocode_response (geocode+)>

  <!ELEMENT geocode (match*)>
  <!ATTLIST geocode id CDATA #REQUIRED
    match_count CDATA #IMPLIED
  >

  <!ELEMENT match (output_address)>
  <!ATTLIST match sequence CDATA #REQUIRED
    longitude CDATA #REQUIRED
    latitude CDATA #REQUIRED
    match_code CDATA #REQUIRED
    error_message CDATA #IMPLIED >

  <!ELEMENT output_address EMPTY>
  <!ATTLIST output_address
    name CDATA #IMPLIED
    house_number CDATA #IMPLIED
    street CDATA #IMPLIED
    builtup_area CDATA #IMPLIED
    order1_area CDATA #IMPLIED
    order8_area CDATA #IMPLIED
    country CDATA #IMPLIED
    postal_code CDATA #IMPLIED
    postal_addon_code CDATA #IMPLIED
    side CDATA #IMPLIED
    percent CDATA #IMPLIED
    edge_id CDATA #IMPLIED>

```

C.3 Data Structures Used by the Routing Engine

Each database user of the routing engine must have the following tables in its schema:

- EDGE
- NODE
- PARTITION
- SIGN_POST

The EDGE and NODE tables store edge and node information about the street network used by the routing engine. To understand how edges and nodes are used to represent street segments, intersections, and other entities in a street network, you must be familiar with the Oracle Spatial network data model, which is described in *Oracle Spatial Topology and Network Data Models*.

The following sections describe the tables used by the routing engine, in alphabetical order by table name.

C.3.1 EDGE Table

The EDGE table contains one row for each directed edge in a street network. Each street segment (a part of a road between two nodes) is an undirected edge that corresponds to one or more directed edges in the EDGE table. The EDGE table contains the columns shown in [Table C-1](#).

Table C-1 EDGE Table

Column Name	Data Type	Description
EDGE_ID	NUMBER	Edge ID number.
START_NODE_ID	NUMBER	Node ID number of the start node of this edge.
END_NODE_ID	NUMBER	Node ID number of the end node of this edge.
PARTITION_ID	NUMBER	Partition ID number of the network partition that contains this edge .
FUNC_CLASS	NUMBER	Functional road class: a number from 1 through 5, with 1 indicating a large, high-speed, high-volume road, and each successive class generally smaller in size, speed, and volume. Class 2 roads have consistent speeds and are used to get traffic to and from class 1 roads. Class 3 roads have high volume and are used to connect class 2 roads. Class 4 roads move volumes of traffic between neighborhoods (for example, a busy main road in a city). Class 5 roads are all other roads (for example, a small, low-volume street in a neighborhood).
LENGTH	NUMBER	Length of this edge, in meters.
SPEED_LIMIT	NUMBER	Assigned speed limit for this edge, in meters per second.
GEOMETRY	SDO_GEOMETRY	Line string geometry representing this edge, with the coordinates ordered from the start node to the end node.
NAME	VARCHAR2(128)	Name of this edge.
DIVIDER	VARCHAR2(1)	A value of N indicates that the edge is not divided; other values indicate whether, where, and how turns are allowed on the divided edge. (The routing engine currently considers only whether the edge is divided or not.)

C.3.2 NODE Table

The NODE table contains one row for each node that is the start node or end node of one or more edges in the street network. A node often corresponds to an intersection (the intersection of two edges); however, a node can be independent of any intersection (for example, the end of a "dead end" or "no outlet" street). The NODE table contains the columns shown in [Table C-2](#).

Table C-2 NODE Table

Column Name	Data Type	Description
NODE_ID	NUMBER	Node ID number.
GEOMETRY	SDO_ GEOMETRY	Point geometry representing this node.
PARTITION_ID	NUMBER	Partition ID number of the network partition that contains this node .

C.3.3 PARTITION Table

The PARTITION table is generated by Oracle based on the contents of the EDGE and NODE tables. (If the contents of the EDGE or NODE table, or both tables, change, you can call the SDO_ROUTER_PARTITION.PARTITION_ROUTER PL/SQL procedure to partition the data, and then swap the new partition table for the existing partition table.) The PARTITION table contains the columns shown in [Table C-3](#).

Table C-3 PARTITION Table

Column Name	Data Type	Description
PARTITION_ID	NUMBER	Partition ID number.
SUBNETWORK	BLOB	Part of the network included in this partition.
NUM_NODES	NUMBER	Number of nodes in this partition.
NUM_NON_ BOUNDARY_EDGES	NUMBER	Number of edges in this partition that are edges that are completely contained within the partition.
NUM_OUTGOING_ BOUNDARY_EDGES	NUMBER	Number of edges in this partition that start in this partition and terminate in another partition. (An edge cannot be in more than two partitions; for example, an edge cannot start in one partition, go through a second partition, and end in a third partition.)
NUM_INCOMING_ BOUNDARY_EDGES	NUMBER	Number of edges in this partition that start in another partition and terminate in this partition. (An edge cannot be in more than two partitions; for example, an edge cannot start in one partition, go through a second partition, and end in a third partition.)

C.3.4 SIGN_POST Table

The SIGN_POST table stores sign information that is used to generate driving directions. For example, a sign might indicate that Exit 33A on US Route 3 South goes toward Winchester. A SIGN_POST row might correspond to a physical sign at an exit ramp on a highway, but it does not need to correspond to a physical sign. The SIGN_POST table contains the columns shown in [Table C-4](#).

Table C-4 SIGN_POST Table

Column Name	Data Type	Description
FROM_EDGE_ID	NUMBER	Edge ID number of the edge to which this sign applies (for example, the street segment containing the exit ramp).
TO_EDGE_ID	NUMBER	Edge ID number of the edge to which this sign points (for example, the street segment to which the exit ramp leads).
RAMP	VARCHAR2(64)	Ramp text (for example, US-3 SOUTH).
EXIT	VARCHAR2(8)	Exit number (for example, 33A).
TOWARD	VARCHAR2(64)	Text indicating where the exit is heading (for example, WINCHESTER).

Complex Spatial Queries: Examples

This appendix provides examples, with explanations, of queries that are more complex than the examples in the reference chapters in [Part II, "Reference Information"](#). This appendix focuses on operators that are frequently used in Spatial applications, such as [SDO_WITHIN_DISTANCE](#) and [SDO_NN](#).

This appendix is based on input from Oracle personnel who provide support and training to Spatial users. The Oracle Spatial training course covers many of these examples, and provides additional examples and explanations.

Before you use any of the examples in this appendix, be sure you understand the usage and reference information for the relevant operator or function in [Part I, "Conceptual and Usage Information"](#) and [Part II, "Reference Information"](#).

This appendix contains the following major sections:

- [Section D.1, "Tables Used in the Examples"](#)
- [Section D.2, "SDO_WITHIN_DISTANCE Examples"](#)
- [Section D.3, "SDO_NN Examples"](#)
- [Section D.4, "SDO_AGGR_UNION Example"](#)

D.1 Tables Used in the Examples

The examples in this appendix refer to tables named `GEOD_CITIES`, `GEOD_COUNTIES`, and `GEOD_INTERSTATES`, which are defined as follows:

```
CREATE TABLE GEOD_CITIES (
  LOCATION      SDO_GEOMETRY,
  CITY          VARCHAR2 (42) ,
  STATE_ABRV    VARCHAR2 (2) ,
  POP90         NUMBER,
  RANK90        NUMBER) ;

CREATE TABLE GEOD_COUNTIES (
  COUNTY_NAME   VARCHAR2 (40) ,
  STATE_ABRV    VARCHAR2 (2) ,
  GEOM          SDO_GEOMETRY) ;

CREATE TABLE GEOD_INTERSTATES (
  HIGHWAY       VARCHAR2 (35) ,
  GEOM          SDO_GEOMETRY) ;
```

D.2 SDO_WITHIN_DISTANCE Examples

The `SDO_WITHIN_DISTANCE` operator identifies the set of spatial objects that are within some specified distance of a given object. You can indicate that the distance is approximate or exact. If you specify `querytype=FILTER`, the distance is approximate because only a primary filter operation is performed; otherwise, the distance is exact because both primary and secondary filtering operations are performed.

[Example D-1](#) finds all cities within 15 miles of the interstate highway I170.

Example D-1 Finding All Cities Within a Distance of a Highway

```
SELECT /*+ ORDERED */ c.city
FROM geod_interstates i, geod_cities c
WHERE i.highway = 'I170'
      AND sdo_within_distance (
          c.location, i.geom,
          'distance=15 unit=mile') = 'TRUE';
```

[Example D-1](#) finds all cities within 15 miles ('distance=15 unit=mile') of the specified highway (`i.highway = 'I170'`), and by default the result is exact (because the `querytype` parameter was not used to limit the query to a primary filter operation). In the `WHERE` clause of this example:

- `i.highway` refers to the `HIGHWAY` column of the `INTERSTATES` table, and `I170` is a value from the `HIGHWAY` column.
- `c.location` specifies the search column (`geometry1`). This is the `LOCATION` column of the `GEOD_CITIES` table.
- `i.geom` specifies the query window (`aGeom`). This is the spatial geometry in the `GEOM` column of the `GEOD_INTERSTATES` table, in the row whose `HIGHWAY` column contains the value `I170`.

[Example D-2](#) finds all interstate highways within 15 miles of the city of Tampa.

Example D-2 Finding All Highways Within a Distance of a City

```
SELECT /*+ ORDERED */ i.highway
FROM geod_cities c, geod_interstates i
WHERE c.city = 'Tampa'
      AND sdo_within_distance (
          i.geom, c.location,
          'distance=15 unit=mile') = 'TRUE';
```

[Example D-2](#) finds all highways within 15 miles ('distance=15 unit=mile') of the specified city (`c.city = 'Tampa'`), and by default the result is exact (because the `querytype` parameter was not used to limit the query to a primary filter operation). In the `WHERE` clause of this example:

- `c.city` refers to the `CITY` column of the `GEOD_CITIES` table, and `Tampa` is a value from the `CITY` column.
- `i.geom` specifies the search column (`geometry1`). This is the `GEOM` column of the `GEOD_INTERSTATES` table.
- `c.location` specifies the query window (`aGeom`). This is the spatial geometry in the `LOCATION` column of the `GEOD_CITIES` table, in the row whose `CITY` column contains the value `Tampa`.

D.3 SDO_NN Examples

The `SDO_NN` operator determines the nearest neighbor geometries to a geometry. No assumptions should be made about the order of the returned results. If you specify no optional parameters, one nearest neighbor geometry is returned.

If you specify the optional `sdo_num_res` keyword, you can request how many nearest neighbors you want, but no other conditions in the `WHERE` clause are evaluated, and any `sdo_batch_size` specification is ignored. For example, assume that you want the five closest banks from an intersection, but only where the bank name is `CHASE`. If the five closest banks are not named `CHASE`, `SDO_NN` with any `sdo_num_res` value and with `sdo_batch_size=5` will return no rows because the `sdo_num_res` keyword only takes proximity into account, and not any `sdo_batch_size` specification or conditions in the `WHERE` clause.

If you specify the optional `sdo_batch_size` keyword instead of the `sdo_num_res` keyword, `SDO_NN` keeps returning neighbor geometries in distance order to the `WHERE` clause. If the `WHERE` clause specifies `bank_name = 'CHASE' AND rownum < 6`, you can return the five closest banks with `bank_name = 'CHASE'`.

`SDO_NN_DISTANCE` is an ancillary operator to the `SDO_NN` operator. It returns the distance of an object returned by the `SDO_NN` operator and is valid only within a call to the `SDO_NN` operator.

Example D-3 finds the five cities nearest to the interstate highway I170 and the distance in miles from the highway for each city, ordered by distance in miles.

Example D-3 Finding the Cities Nearest to a Highway

```
SELECT /*+ ORDERED */
       c.city,
       sdo_nn_distance (1) distance_in_miles
FROM   geod_interstates i,
       geod_cities c
WHERE  i.highway = 'I170'
       AND sdo_nn(c.location, i.geom,
                  'sdo_num_res=5 unit=mile', 1) = 'TRUE'
ORDER BY distance_in_miles;
```

In **Example D-3**, because the `/*+ ORDERED*/` optimizer hint is used, it is important to have an index on the `GEOD_INTERSTATES.HIGHWAY` column. In this example, the hint forces the query to locate highway I170 before it tries to find nearest neighbor geometries. In the `WHERE` clause of this example:

- `i.highway` refers to the `HIGHWAY` column of the `GEOD_INTERSTATES` table, and `I170` is a value from the `HIGHWAY` column.
- `c.location` specifies the search column (`geometry1`). This is the `LOCATION` column of the `GEOD_CITIES` table.
- `i.geom` specifies the query window (`geometry2`). This is the spatial geometry in the `GEOM` column of the `GEOD_INTERSTATES` table, in the row whose `HIGHWAY` column contains the value `I170`.
- `sdo_num_res=5` specifies how many nearest neighbor geometries to find.
- `unit=mile` specifies the unit of measurement to associate with distances returned by the `SDO_NN_DISTANCE` ancillary operator.
- `1` (in `sdo_nn_distance (1)` and `'sdo_num_res=5 unit=mile', 1`) is the number parameter value that associates the call to `SDO_NN` with the call to `SDO_NN_DISTANCE`.

In [Example D-3](#), `ORDER BY distance_in_miles` orders the results from the `WHERE` clause by distance in miles.

The statement in [Example D-3](#) produces the following output (slightly reformatted for readability):

CITY	DISTANCE_IN_MILES
St Louis	5.36297295
Springfield	78.7997464
Peoria	141.478022
Evansville	158.22422
Springfield	188.508631

[Example D-4](#) extends [Example D-3](#) by limiting the results to cities with a 1990 population over a certain number. It finds the five cities nearest to the interstate highway I170 that have a population greater than 300,000, the 1990 population for each city, and the distance in miles from the highway for each city, ordered by distance in miles.

Example D-4 Finding the Cities Above a Specified Population Nearest to a Highway

```
SELECT /*+ ORDERED NO_INDEX(c pop90_idx) */
       c.city, pop90,
       sdo_nn_distance (1) distance_in_miles
FROM   geod_interstates i,
       geod_cities c
WHERE  i.highway = 'I170'
       AND sdo_nn(c.location, i.geom,
                  'sdo_batch_size=10 unit=mile', 1) = 'TRUE'
       AND c.pop90 > 300000
       AND rownum < 6
ORDER BY distance_in_miles;
```

In [Example D-4](#), because the `ORDERED` optimizer hint is used, it is important to have an index on the `GEOD_INTERSTATES.HIGHWAY` column. In this example, the hint forces the query to locate highway I170 before it tries to find nearest neighbor geometries.

To ensure correct results, disable all nonspatial indexes on columns that come from the same table as the `SDO_NN` search column (`geometry1`). In this example, the `NO_INDEX(c pop90_idx)` optimizer hint disables the nonspatial index on the `POP90` column.

In the `WHERE` clause of this example:

- `sdo_batch_size=10` causes geometries to be returned continually (in distance order, in batches of 10 geometries), to be checked to see if they satisfy the other conditions in the `WHERE` clause.
- `c.pop90 > 300000` restricts the results to rows where the `POP90` column value is greater than 300000.
- `rownum < 6` limits the number of results returned to five.

In [Example D-4](#), `ORDER BY distance_in_miles` orders the results from the `WHERE` clause by distance in miles.

The statement in [Example D-4](#) produces the following output (slightly reformatted for readability):

CITY	POP90	DISTANCE_IN_MILES
------	-------	-------------------

```

-----
St Louis          396685      5.36297295
Kansas City      435146      227.404883
Indianapolis     741952      234.708666
Memphis          610337      244.202072
Chicago          2783726     253.547961

```

D.4 SDO_AGGR_UNION Example

When you use the [SDO_AGGR_UNION](#) aggregate function, very large geometries can result. When geometries have many coordinates, spatial operations (such as union) can be time-consuming. It may be better to divide a single spatial aggregate union operation function into multiple nested aggregate functions in the same SQL statement.

[Example D-5](#) aggregates all the counties in Texas, producing the boundary for the state of Texas.

Example D-5 Performing Aggregate Union of All Counties in Texas

```

SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,0.5)) aggr_geom
FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,0.5)) aggr_geom
      FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,0.5)) aggr_geom
            FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(aggr_geom,0.5)) aggr_geom
                  FROM (SELECT sdo_aggr_union(mdsys.sdoaggrtype(geom,0.5)) aggr_geom
                        FROM geod_counties WHERE state_abrv='TX'
                        GROUP BY mod(rownum,16)
                  )
            )
            GROUP BY mod (rownum, 8)
      )
      GROUP BY mod (rownum, 4)
    )
    GROUP BY mod (rownum, 2)
);

```

Glossary

area

An extent or region of dimensional space.

attribute

Descriptive information characterizing a geographical feature such as a point, line, or area.

attribute data

Nondimensional data that provides additional descriptive information about multidimensional data, for example, a class or feature such as a bridge or a road.

batch geocoding

An operation that simultaneously geocodes many records from one table. *See also* [geocoding](#).

boundary

1. The lower or upper extent of the range of a dimension, expressed by a numeric value.
2. The line representing the outline of a polygon.

Cartesian coordinate system

A coordinate system in which the location of a point in n -dimensional space is defined by distances from the point to the reference plane. Distances are measured parallel to the planes intersecting a given reference plane. *See also* [coordinate system](#).

colocation

The presence of two or more spatial objects at the same location or at significantly close distances from each other.

contain

A geometric relationship where one object encompasses another and the inner object does not touch any boundaries of the outer. The outer object *contains* the inner object. *See also* [inside](#).

convex hull

A simple convex polygon that completely encloses the associated geometry object.

coordinate

A set of values uniquely defining a point in an n -dimensional coordinate system.

coordinate reference system

Synonymous with [coordinate system](#) in Oracle Spatial documentation. The term *coordinate reference system* is used extensively by the European Petroleum Survey Group (EPSG).

coordinate system

A reference system for the unique definition for the location of a point in *n*-dimensional space. Also called a spatial reference system. *See also* [Cartesian coordinate system](#), [geodetic coordinates](#), [projected coordinates](#), and [local coordinates](#).

cover

A geometric relationship in which one object encompasses another and the inner object touches the boundary of the outer object in one or more places.

data dictionary

A repository of information about data. A data dictionary stores relational information on all objects in a database.

datum transformation

See [transformation](#).

dimensional data

Data that has one or more dimensional components and is described by multiple values.

direction

The direction of an LRS geometric segment is indicated from the start point of the geometric segment to the end point. Measures of points on a geometric segment always increase along the direction of the geometric segment.

disjoint

A geometric relationship where two objects do not interact in any way. Two *disjoint* objects do not share any element or piece of their geometry.

element

A basic building block (point, line string, or polygon) of a geometry.

equal

A geometric relationship in which two objects are considered to represent the same geometric figure. The two objects must be composed of the same number of points; however, the ordering of the points defining geometries of the two objects may differ (clockwise or counterclockwise).

extent

A rectangle bounding a map, the size of which is determined by the minimum and maximum map coordinates.

feature

An object with a distinct set of characteristics in a spatial database.

geocoding

The process of converting tables of address data into standardized address, location, and possibly other data. *See also* [batch geocoding](#).

geodetic coordinates

Angular coordinates (longitude and latitude) closely related to spherical polar coordinates and defined relative to a particular Earth geodetic datum. Also referred to as geographic coordinates.

geodetic datum

A means of representing the figure of the Earth, usually as an oblate ellipsoid of revolution, that approximates the surface of the Earth locally or globally, and is the reference for the system of geodetic coordinates.

geographic coordinates

See [geodetic coordinates](#).

geographic information system (GIS)

A computerized database management system used for the capture, conversion, storage, retrieval, analysis, and display of spatial data.

geographically referenced data

See [spatiotemporal data](#).

geometric segment (LRS segment)

An LRS element that contains start and end measures for its start and end points, and that can contain measures for other points on the segment.

geometry

The geometric representation of the shape of a spatial feature in some coordinate space. A geometry is an ordered sequence of vertices that are connected by straight line segments or circular arcs.

georeferenced data

See [spatiotemporal data](#).

GIS

See [geographic information system \(GIS\)](#).

grid

A data structure composed of points located at the nodes of an imaginary grid. The spacing of the nodes is constant in both the horizontal and vertical directions.

hole

A subelement of a polygon that negates a section of its interior. For example, consider a polygon representing a map of buildable land with an inner polygon (a hole) representing where a lake is located.

homogeneous

Spatial data of one feature type such as points, lines, or regions.

hyperspatial data

In mathematics, any space having more than the three standard X, Y, and Z dimensions. Sometimes referred to as multidimensional data.

index

A database object that is used for fast and efficient access to stored information.

inside

A geometric relationship where one object is surrounded by a larger object and the inner object does not touch the boundary of the outer. The smaller object is *inside* the larger. *See also* [contain](#).

key

A field in a database used to obtain access to stored information.

latitude

North/south position of a point on the Earth defined as the angle between the normal to the Earth's surface at that point and the plane of the equator.

layer

A collection of geometries having the same attribute set and stored in a geometry column.

line

A geometric object represented by a series of points, or inferred as existing between two coordinate points.

line string

One or more pairs of points that define a line segment. *See also* [multiline string](#).

linear feature

Any spatial object that can be treated as a logical set of linear segments.

local coordinates

Cartesian coordinates in a non-Earth (non-georeferenced) coordinate system.

longitude

East/west position of a point on the Earth defined as the angle between the plane of a reference meridian and the plane of a meridian passing through an arbitrary point.

LRS point

A point with linear measure information along a geometric segment. *See also* [geometric segment \(LRS segment\)](#).

measure

The linear distance (in the LRS measure dimension) to a point measured from the start point (for increasing values) or end point (for decreasing values) of the geometric segment.

measure range

The measure values at the start and end of a geometric segment.

minimum bounding rectangle (MBR)

A single rectangle that minimally encloses a geometry or a collection of geometries.

multidimensional data

See [hyperspatial data](#).

multiline string

A geometry object made up of nonconnected line string elements (for example, a street with a gap caused by a city park, such as Sixth Avenue in New York City with Central Park as the gap). *See also* [line string](#).

multipolygon

A polygon collection geometry in which rings must be grouped by polygon, and the first ring of each polygon must be the exterior ring.

neighborhood influence

See [spatial correlation](#).

offset

The perpendicular distance between a point along a geometric segment and the geometric segment. Offsets are positive if the points are on the left side along the segment direction and are negative if they are on the right side. Points are on a geometric segment if their offsets to the segment are zero.

oriented point

A special type of point geometry that includes coordinates representing the locations of the point and a virtual end point, to indicate an orientation vector that can be used for rotating a symbol at the point or extending a label from the point

polygon

A class of spatial objects having a nonzero area and perimeter, and representing a closed boundary region of uniform characteristics.

primary filter

The operation that permits fast selection of candidate records to pass along to the secondary filter. The primary filter compares geometry approximations to reduce computation complexity and is considered a lower-cost filter. Because the primary filter compares geometric approximations, it returns a superset of the exact result set. *See also* [secondary filter](#) and [two-tier query model](#).

projected coordinates

Planar Cartesian coordinates that result from performing a mathematical mapping from a point on the Earth's surface to a plane. There are many such mathematical mappings, each used for a particular purpose.

projection

The point on the LRS geometric segment with the minimum distance to the specified point.

proximity

A measure of distance between objects.

query

A set of conditions or questions that form the basis for the retrieval of information from a database.

query window

Area within which the retrieval of spatial information and related attributes is performed.

RDBMS

See [Relational Database Management System \(RDBMS\)](#).

recursion

A process, function, or routine that executes continuously until a specified condition is met.

region

An extent or area of multidimensional space.

Relational Database Management System (RDBMS)

A computer program designed to store and retrieve shared data. In a relational system, data is stored in tables consisting of one or more rows, each containing the same set of columns. Oracle Database is an object-relational database management system. Other types of database systems are called hierarchical or network database systems.

resolution

The number of subdivision levels of data.

scale

The ratio of the distance on a map, photograph, or image to the corresponding image on the ground, all expressed in the same units.

secondary filter

The operation that applies exact computations to geometries that result from the primary filter. The secondary filter yields an accurate answer to a spatial query. The secondary filter operation is computationally expensive, but it is only applied to the primary filter results, not the entire data set. See also [primary filter](#) and [two-tier query model](#).

shape points

Points that are specified when an LRS segment is constructed, and that are assigned measure information.

sort

The operation of arranging a set of items according to a key that determines the sequence and precedence of items.

spatial

A generic term used to reference the mathematical concept of n -dimensional data.

spatial binning

The process of discretizing the location values into a small number of groups associated with geographical areas. Also referred to as *spatial discretization*.

spatial correlation

The phenomenon of the location of a specific object in an area affecting some nonspatial attribute of the object. Also referred to as *neighborhood influence*.

spatial data

Data that is referenced by its location in n -dimensional space. The position of spatial data is described by multiple values. See also [hyperspatial data](#).

spatial data model

A model of how objects are located on a spatial context.

spatial data structures

A class of data structures designed to store spatial information and facilitate its manipulation.

spatial database

A database containing information indexed by location.

spatial discretization

See [spatial binning](#).

spatial join

A query in which each of the geometries in one layer is compared with each of the geometries in the other layer. Comparable to a spatial cross product.

spatial query

A query that includes criteria for which selected features must meet location conditions.

spatial reference system

See [coordinate system](#).

spatiotemporal data

Data that contains time or location (or both) components as one of its dimensions, also referred to as geographically referenced data or georeferenced data.

SQL*Loader

A utility to load formatted data into spatial tables.

tolerance

The distance that two points can be apart and still be considered the same (for example, to accommodate rounding errors). The tolerance value must be a positive number greater than zero. The significance of the value depends on whether or not the spatial data is associated with a geodetic coordinate system.

touch

A geometric relationship where two objects share a common point on their boundaries, but their interiors do not intersect.

transformation

The conversion of coordinates from one coordinate system to another coordinate system. If the coordinate system is georeferenced, transformation can involve datum transformation: the conversion of geodetic coordinates from one geodetic datum to another geodetic datum, usually involving changes in the shape, orientation, and center position of the reference ellipsoid.

two-tier query model

The query model used by Spatial to resolve spatial queries and spatial joins. Two distinct filtering operations (primary and secondary) are performed to resolve queries. The output of both operations yields the exact result set. See also [primary filter](#) and [secondary filter](#).

Symbols

, 6-32, 6-33

Numerics

0

SRID value used with SDO_CS.VIEWPORT_TRANSFORM function, 13-48
type 0 (zero) element, 2-23

10g

upgrading spatial data to Oracle Database
10g, 17-2

3D

formats of LRS functions, 7-7
not supported with geodetic data, 6-40

A

ADD_PREFERENCE_FOR_OP procedure, 13-4

addresses

representing for geocoding, 5-1

aggregate functions

description, 1-14

reference information, 12-1

SDO_AGGR_CENTROID, 12-2

SDO_AGGR_CONCAT_LINES, 12-3

SDO_AGGR_CONVEXHULL, 12-5

SDO_AGGR_LRS_CONCAT, 12-6

SDO_AGGR_MBR, 12-8

SDO_AGGR_UNION, 12-9

SDOAGGRTYPE object type, 1-14

AGGREGATES_FOR_GEOMETRY function, 18-2

AGGREGATES_FOR_LAYER function, 18-4

aliases, 6-9

ALL_SDO_GEOM_METADATA view, 2-29

ALL_SDO_INDEX_INFO view, 2-32

ALL_SDO_INDEX_METADATA view, 2-32

ALTER INDEX statement, 10-2

REBUILD clause, 10-4

RENAME TO clause, 10-7

angle units, 6-29

ANYINTERACT

SDO_ANYINTERACT operator, 11-3

topological relationship, 1-12

APPEND function, 20-3

application size (hardware) requirements, 1-17

arc

densifying, 15-6

not supported with geodetic data, 6-5

area, 15-8

area units, 6-29

average minimum bounding rectangle, 19-2

AVERAGE_MBR procedure, 19-2

B

batch route requests, C-16

DTD, C-16

example, C-13

batch route responses

DTD, C-17

example, C-13

batch_route_request element, C-16

bearing

point at, 20-22

BIN_GEOMETRY function, 18-6

BIN_LAYER procedure, 18-8

binning

spatial, 8-3

See also bins

bins

assigning, 18-8

computing, 18-6

tiled, 18-20

boundary

of area, 1-11

of line string, 1-11

of multiline string, 1-11

of polygon, 1-11

bounding rectangle

minimum, 19-7

buffer area, 15-10

bulk loading of spatial data, 3-1

C

C language

examples (using OCI), 1-17

Cartesian coordinates, 1-5, 6-2

center of gravity (centroid), 15-13

centroid

- SDO_AGGR_CENTROID aggregate function, 12-2
- SDO_CENTROID function, 15-13
- circle
 - creating polygon approximating, 20-4
 - not supported with geodetic data, 6-5
 - type, 2-10
- CIRCLE_POLYGON function, 20-4
- CLIP_GEOM_SEGMENT function, 16-5
- clipping
 - geometric segment, 7-9
- COLOCATED_REFERENCE_FEATURES procedure, 18-10
- colocation mining, 8-4
- column name
 - restrictions on spatial column names, 2-30
- COLUMN_NAME (in USER_SDO_GEOM_METADATA), 2-30
- compatibility, A-1
- complex examples
 - queries, D-1
- compound element, 2-8
- compound line string, 2-10, 2-17
- compound polygon, 2-10
- CONCAT_LINES function, 20-6
- CONCATENATE_GEOM_SEGMENTS function, 16-7
- concatenating
 - geometric segments, 7-10
 - line or multiline geometries, 12-3
 - LRS geometries, 7-11, 12-6
- CONNECTED_GEOM_SEGMENTS function, 16-10
- consistency
 - checking for valid geometry types, 15-37
- constraining data to a geometry type, 4-2
- constructors
 - SDO_GEOMETRY object type, 2-13
- CONTAINS
 - SDO_CONTAINS operator, 11-5
 - topological relationship, 1-12
- CONVERSION_FACTOR column
 - in SDO_ANGLE_UNITS table, 6-29
 - in SDO_AREA_UNITS table, 6-30
 - in SDO_DIST_UNITS table, 6-32
- CONVERT_NADCON_TO_XML procedure, 13-6
- CONVERT_NTV2_TO_XML procedure, 13-8
- CONVERT_TO_LRS_DIM_ARRAY function, 16-12
- CONVERT_TO_LRS_GEOM function, 16-14
- CONVERT_TO_LRS_LAYER function, 16-16
- CONVERT_TO_STD_DIM_ARRAY function, 16-18
- CONVERT_TO_STD_GEOM function, 16-19
- CONVERT_TO_STD_LAYER function, 16-20
- CONVERT_UNIT function, 20-8
- CONVERT_XML_TO_NADCON procedure, 13-10
- CONVERT_XML_TO_NTV2 procedure, 13-12
- converting
 - geometric segments
 - overview, 7-14
 - subprograms for, 16-3
- convex hull
 - SDO_AGGR_CONVEXHULL aggregate function, 12-5
 - SDO_CONVEXHULL function, 15-15
- coordinate dimension
 - ST_CoordDim method, 2-12
- coordinate reference systems
 - See coordinate systems
- coordinate systems, 1-5
 - conceptual and usage information, 6-1
 - data structures supplied by Oracle, 6-8
 - example, 6-41
 - local, 6-6
 - subprogram reference information, 13-1
 - unit of measurement support, 2-35
 - user-defined, 6-34
- coordinates
 - Cartesian, 1-5, 6-2
 - geodetic, 1-6, 6-2
 - geographic, 1-6, 6-2
 - local, 1-6, 6-2
 - projected, 1-6, 6-2
- COVEREDBY
 - SDO_COVEREDBY operator, 11-6
 - topological relationship, 1-12
- COVERS
 - SDO_COVERS operator, 11-7
 - topological relationship, 1-12
- CPU requirements for applications using Spatial, 1-17
- CREATE INDEX statement, 10-8
- CREATE_CONCATENATED_OP procedure, 13-14
- CREATE_OBVIOUS_EPSG_RULES procedure, 13-15
- CREATE_PREF_CONCATENATED_OP procedure, 13-16
- creating
 - geometric segments
 - subprograms for, 16-1
- cross-schema index creation, 4-3
- CS_SRS table, 6-26
- current release
 - upgrading spatial data to, 17-2
- cutoff_distance attribute
 - of batch route request, C-17

D

- data mining
 - spatial
 - colocation mining, 8-4
 - conceptual and usage information, 8-1
 - function reference information, 18-1
- data model, 1-4
 - LRS, 7-6
- data types
 - geocoding, 5-4
 - spatial, 2-1
- database links
 - not supported if spatial index is defined on the table, 4-6

- datum
 - geodetic, 1-6, 6-2
 - MDSYS.SDO_DATUMS_OLD_FORMAT table, 6-30
 - MDSYS.SDO_DATUMS_OLD_SNAPSHOT table, 6-30
 - transformation, 6-2
- dblink
 - not supported if spatial index is defined on the table, 4-6
- DEFINE_GEOM_SEGMENT procedure, 16-22
- defining
 - geometric segment, 7-8
- DELETE_ALL_EPSG_RULES procedure, 13-18
- DELETE_OP procedure, 13-19
- densification of arcs, 15-6
- DETERMINE_CHAIN function, 13-20
- DETERMINE_DEFAULT_CHAIN function, 13-22
- difference
 - SDO_GEOM.SDO_DIFFERENCE function, 15-17
- dimension (in SDO_GTYPE), 2-5, 2-6
 - Get_Dims method, 2-11
 - Get_LRS_Dim method, 2-11
- DIMINFO (in USER_SDO_GEOM_METADATA), 2-30
- direction of geometric segment, 7-3
 - concatenation result, 7-11
- discretization (binning)
 - spatial, 8-3
 - See also* bins
- DISJOINT
 - topological relationship, 1-12
- disk storage requirements for applications using Spatial, 1-17
- distance
 - SDO_NN_DISTANCE ancillary operator, 11-20
 - WITHIN_DISTANCE function, 15-43
- distance units, 6-32
- distance_unit attribute
 - of route request, C-11
- distributed transactions
 - requirements to ensure spatial index consistency, 4-6
- Douglas-Peucker algorithm for geometry
 - simplification, 20-30
- downgrading
 - Oracle Spatial to the previous Oracle Database release, A-1
- driving directions
 - in route request, C-11
- DROP INDEX statement, 10-12
- duplicate vertices
 - removing, 20-27
- dynamic query window, 4-7
- DYNAMIC_SEGMENT function, 16-25
- E**

- EDGE table
 - routing engine use of, C-20
- editing
 - geometric segments
 - subprograms for, 16-1
- ELEM_INFO (SDO_ELEM_INFO attribute), 2-7
- elements, 1-4
 - extracting from a geometry, 20-11
 - returning number of elements in a geometry, 20-17
 - returning number of vertices in a geometry, 20-18
- ellipse
 - creating polygon approximating, 20-9
- ELLIPSE_POLYGON function, 20-9
- ellipsoids
 - MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT table, 6-32
 - MDSYS.SDO_ELLIPSOIDS_OLD_SNAPSHOT table, 6-32
- embedded SDO_GEOMETRY object in user-defined type, 9-1
- end location for route, C-10
- EPSG, 6-8
- EPSG SRID
 - mapping Oracle SRID to, 13-30
 - mapping to Oracle SRID, 13-29
- EQUAL
 - SDO_EQUAL operator, 11-8
 - topological relationship, 1-12
- error messages
 - geocoding, 5-4
 - Spatial, 1-17
- ESTIMATE_RTREE_INDEX_SIZE function, 19-4
- ETYPE (SDO_ETYPE value), 2-8, 2-9
- European Petroleum Survey Group
 - See* EPSG
- examples
 - batch route request and response, C-13
 - C, 1-17
 - complex queries, D-1
 - coordinate systems, 6-41
 - creating, indexing, and querying spatial data, 2-1
 - directory for Spatial examples, 1-17
 - linear referencing system (LRS), 7-15
 - many geometry types (creating), 2-14
 - OCI (Oracle Call Interface), 1-17
 - PL/SQL, 1-17
 - route request and response, C-5
 - route request with previously geocoded locations, C-8
 - route response with previously geocoded locations, C-9
 - SQL, 1-17
- exchanging partitions including indexes, 4-5
- Export utility
 - with spatial indexes and data, 4-5
- EXTENT_OF function, 19-7
- exterior polygon rings, 2-8, 2-15, 2-17
- EXTRACT function, 20-11

F

- features
 - linear, 7-5
- FILTER
 - SDO_FILTER operator, 11-9
- FILTER mask value for SDO_JOIN, 11-14
- FIND_GEOG_CRS function, 13-23
- FIND_LRS_DIM_POS function, 16-27
- FIND_MEASURE function, 16-28
- FIND_OFFSET function, 16-30
- FIND_PROJ_CRS function, 13-25
- formatted addresses, 5-1
- FROM_815_TO_81X procedure
 - use TO_CURRENT instead, 17-1
- FROM_OGC_SIMPLEFEATURE_SRS
 - function, 13-27
- FROM_USNG function, 13-28
- FROM_WKBGEOMETRY function, 20-13
- FROM_WKTGEOMETRY function, 20-15
- function-based indexes
 - with SDO_GEOMETRY objects, 9-3
- functions
 - spatial aggregate, 12-1
 - supported by approximations with geodetic data, 6-40

G

- GC_AREA_<suffix> table, 5-10
- GC_COUNTRY_PROFILE table, 5-11
- GC_INTERSECTION_<suffix> table, 5-13
- GC_POI_<suffix> table, 5-14
- GC_POSTAL_CODE_<suffix> table, 5-15
- GC_ROAD_<suffix> table, 5-16
- GC_ROAD_SEGMENT_<suffix> table, 5-18
- GEOCODE function, 14-2
- GEOCODE_ADDR function, 14-3
- GEOCODE_ADDR_ALL function, 14-5
- GEOCODE_ALL function, 14-7
- GEOCODE_AS_GEOMETRY function, 14-9
- geocoding, 1-15
 - concepts, 5-1
 - data requirements, 5-7
 - data types for, 5-4
 - error messages, 5-4
 - from a place name, 5-8
 - match codes, 5-3
 - match modes, 5-2
 - reverse, 14-10
 - subprogram reference information, 14-1
 - usage information, 5-1
- geocoding requests
 - DTD, C-18
- geocoding responses
 - DTD, C-19
- geodetic coordinates, 1-6, 6-2
 - arcs and circles not supported, 6-5
 - functions supported by approximations, 6-40
 - support for, 6-2
- geodetic datum, 1-6, 6-2

- geodetic indexes, 4-2
- geodetic MBRs, 6-3
- geographic coordinates
 - See geodetic coordinates
- geography markup language (GML)
 - converting geometry to, 20-33
- GEOM_SEGMENT_END_MEASURE
 - function, 16-32
- GEOM_SEGMENT_END_PT function, 16-33
- GEOM_SEGMENT_LENGTH function, 16-34
- GEOM_SEGMENT_START_MEASURE
 - function, 16-35
- GEOM_SEGMENT_START_PT function, 16-36
- geometric segment
 - clipping, 7-9
 - concatenating, 7-10
 - aggregate, 7-11, 12-6
 - converting (overview), 7-14
 - converting (subprograms for), 16-3
 - creating (subprograms for), 16-1
 - defining, 7-8
 - definition of, 7-2
 - direction, 7-3
 - direction with concatenation, 7-11
 - editing (subprograms for), 16-1
 - locating point on, 7-12
 - offsetting, 7-12
 - projecting point onto, 7-13
 - querying (subprograms for), 16-2
 - redefining, 7-9
 - scaling, 7-11
 - splitting, 7-10
- geometry subprograms
 - reference information, 15-1
- geometry types, 1-3
 - constraining data to, 4-2
 - Get_GType method, 2-11
 - SDO_GTYPE, 2-5
- GeoRaster
 - checks and actions after upgrade, A-1
- Get_Dims method, 2-11
- Get_GType method, 2-11
- Get_LRS_Dim method, 2-11
- GET_MEASURE function, 16-37
- GET_NEXT_SHAPE_PT function, 16-38
- GET_NEXT_SHAPE_PT_MEASURE function, 16-40
- GET_PREV_SHAPE_PT function, 16-42
- GET_PREV_SHAPE_PT_MEASURE function, 16-44
- Get_WKB method, 2-11
- Get_WKT method, 2-12
- GETNUMELEM function, 20-17
- GETNUMVERTICES function, 20-18
- GETVERTICES function, 20-19
- GML (geography markup language)
 - converting geometry to, 20-33
- GTYPE (SDO_GTYPE attribute), 2-5
 - constraining data to a geometry type, 4-2
 - Get_Dims method, 2-11
 - Get_GType method, 2-11
 - GET_LRS_Dim method, 2-11

H

hardware requirements for applications using Spatial, 1-17

I

id attribute

of route request, C-11

Import utility

with spatial indexes and data, 4-5

index

distributed transactions and spatial index consistency, 4-6

indexes

creating, 4-1

cross-schema, 4-3

parallel execution, 10-9

description of Spatial indexing, 1-9

extending spatial indexing capabilities, 9-1

function-based with SDO_GEOMETRY objects, 9-3

geodetic and non-geodetic, 4-2

partitioned, 4-3

exchanging partitions including indexes, 4-5

quadtree

deprecated feature of Spatial, 1-9

rebuilding, 10-4

parallel execution, 10-3, 10-5

R-tree

description, 1-9

requirements before creating, 4-1

size (R-tree), 19-4

index-organized table

cannot create spatial index on, 2-30, 10-10

INITIALIZE_INDEXES_FOR_TTS procedure, 20-21

input_location element, C-12

inserting spatial data

PL/SQL, 3-3

INSIDE

SDO_INSIDE operator, 11-12

topological relationship, 1-12

installation, A-1

interMedia requirement for Locator, B-1

INTEPRETATION (SDO_INTERPRETATION value), 2-8

interaction

ANYINTERACT, 1-12

interior

of an area, 1-11

interior polygon rings, 2-8, 2-15, 2-17

interMedia

proper installation required for Locator, B-1

intersection, 15-21, 16-53

intersections

GC_INTERSECTION_<suffix> table, 5-13

inverse flattening, 6-22, 6-33

IS_GEOM_SEGMENT_DEFINED function, 16-46

IS_MEASURE DECREASING function, 16-47

IS_MEASURE INCREASING function, 16-48

IS_SHAPE_PT_MEASURE function, 16-49

J

Java application programming interface (API) for Spatial, 1-15

join

SDO_JOIN operator, 11-13

L

language attribute

of route request, C-11

layer, 1-5

transforming, 13-38

validating with context, 15-41

layer_gtype

constraining data to a geometry type, 4-2

length

SDO_LENGTH function, 15-23

line

converting polygon to, 20-24

data, 1-5

length, 15-23

line string

boundary of, 1-11

compound, 2-10, 2-17

reversing, 20-29

self-crossing, 1-4

linear features, 7-5

linear measure, 7-3

linear referencing system (LRS)

3D formats of functions, 7-7

not supported with geodetic data, 6-40

conceptual and usage information, 7-1

data model, 7-6

example, 7-15

Get_LRS_Dim method, 2-11

limiting indexing to X and Y dimensions, 7-7

LRS points, 7-5

segments, 7-2

subprogram reference information, 16-1

tolerance values with LRS functions, 7-15

upgrading data to current release, A-1

loading spatial data, 3-1

local coordinate systems, 6-6

local coordinates, 1-6, 6-2

LOCAL partitioning

spatial indexes, 4-3

LOCATE_PT function, 16-51

location prospecting, 8-5

Locator, B-1

LRS

See linear referencing system (LRS)

LRS points, 7-5

LRS_INTERSECTION function, 16-53

M

map projections

MDSYS.SDO_PROJECTIONS_OLD_SNAPSHOT table, 6-33

MAP_EPSG_SRID_TO_ORACLE function, 13-29

MAP_ORACLE_SRID_TO_EPSG function, 13-30
 match codes, 5-3
 match modes, 5-2
 MBR
 See minimum bounding rectangle (MBR)
 MDDATA schema, 1-16
 MDSYS schema, 1-2
 MDSYS.CS_SRS table, 6-26
 MDSYS.SDO_ANGLE_UNITS table, 6-29
 MDSYS.SDO_AREA_UNITS view, 6-29
 MDSYS.SDO_CS package, 13-1
 MDSYS.SDO_DATUMS_OLD_FORMAT table, 6-30
 MDSYS.SDO_DATUMS_OLD_SNAPSHOT table, 6-30
 MDSYS.SDO_DIST_UNITS view, 6-32
 MDSYS.SDO_ELLIPSOIDS_OLD_FORMAT table, 6-32
 MDSYS.SDO_ELLIPSOIDS_OLD_SNAPSHOT table, 6-32
 MDSYS.SDO_GCDR package, 14-1
 MDSYS.SDO_PROJECTIONS_OLD_FORMAT table, 6-33
 MDSYS.SDO_PROJECTIONS_OLD_SNAPSHOT table, 6-33
 MDSYS.SDO_SAM package, 18-1
 measure, 7-3
 populating, 7-3
 resetting, 16-67
 reversing, 16-71
 with multiline strings and polygons with holes, 7-5
 measure range, 7-5
 MEASURE_RANGE function, 16-55
 MEASURE_TO_PERCENTAGE function, 16-56
 messages
 Spatial error messages, 1-17
 methods
 SDO_GEOMETRY object type, 2-11
 migration
 See upgrading
 minimum bounding rectangle (MBR)
 AVERAGE_MBR procedure, 19-2
 EXTENT_OF function, 19-7
 geodetic, 6-3
 SDO_AGGR_MBR aggregate function, 12-8
 SDO_MAX_MBR_ORDINATE function, 15-25
 SDO_MBR function, 15-27
 SDO_MIN_MBR_ORDINATE function, 15-29
 mining
 See data mining
 MIX_INFO procedure, 19-8
 multiline string
 boundary of, 1-11
 LRS measure values, 7-5
 multimaster replication
 SDO_GEOMETRY objects, B-3
 multipolygon, 2-17

N

NaC coordinate reference system, 6-40
 NADCON grids
 converting Spatial XML format to NADCON format, 13-10
 converting to Spatial XML format, 13-6
 naming considerations
 Spatial table and column names, 2-30
 nearest neighbor
 SDO_NN operator, 11-16
 SDO_NN_DISTANCE operator, 11-20
 neighborhood influence, 8-3
 NLS_LENGTH_SEMANTICS parameter
 spatial queries on UTF8 data, 4-6
 NODE table
 routing engine use of, C-21
 non-geodetic indexes, 4-2
 NTv2 grids
 converting Spatial XML format to NTv2 format, 13-12
 converting to Spatial XML format, 13-8

O

object types
 embedding SDO_GEOMETRY objects in, 9-1, 9-4
 OCG (Open Geospatial Consortium) simple features conformance, 1-16
 OCI (Oracle Call Interface) examples, 1-17
 ODM
 See Oracle Data Mining (ODM)
 offset, 7-3
 FIND_OFFSET function, 16-30
 OFFSET_GEOM_SEGMENT function, 16-58
 offsetting
 geometric segment, 7-12
 ON
 SDO_ON operator, 11-22
 topological relationship, 1-12
 Open Geospatial Consortium (OCG) simple features conformance, 1-16
 operators
 cross-schema invocation, 4-12
 ORDERED optimizer hint with, 1-14
 SDO_FILTER, 11-10
 SDO_JOIN, 11-14
 overview, 1-13
 performance-related guidelines, 1-13
 SDO_ANYINTERACT, 11-3
 SDO_CONTAINS, 11-5
 SDO_COVEREDBY, 11-6
 SDO_COVERS, 11-7
 SDO_EQUAL, 11-8
 SDO_FILTER, 11-9
 SDO_INSIDE, 11-12
 SDO_JOIN, 11-13
 SDO_NN, 11-16
 SDO_NN_DISTANCE, 11-20
 SDO_ON, 11-22
 SDO_OVERLAPBDYDISJOINT, 11-23

- SDO_OVERLAPBDYINTERSECT, 11-25
- SDO_OVERLAPS, 11-27
- SDO_RELATE, 11-29
- SDO_TOUCH, 11-33
- SDO_WITHIN_DISTANCE, 11-35
- optimized rectangle, 2-10
- optimizer hint (ORDERED) with spatial operators, 1-14
 - SDO_FILTER, 11-10
 - SDO_JOIN, 11-14
- Oracle Call Interface (OCI) examples, 1-17
- Oracle Data Mining (ODM)
 - spatial analysis and mining information, 8-1
- Oracle Database 10g
 - upgrading Spatial to, A-1
- Oracle Locator
 - See* Locator
- ORDERED optimizer hint with spatial operators, 1-14
 - SDO_FILTER, 11-10
 - SDO_JOIN, 11-14
- oriented point
 - illustration and example, 2-21
- OVERLAPBDYDISJOINT
 - SDO_OVERLAPBDYDISJOINT operator, 11-23
 - topological relationship, 1-12
- OVERLAPBDYINTERSECT
 - SDO_OVERLAPBDYINTERSECT operator, 11-25
 - topological relationship, 1-12
- OVERLAPS
 - SDO_OVERLAPS operator, 11-27

P

- parallel execution for index creation and rebuilding, 10-3, 10-5, 10-9
- PARTITION table
 - routing engine use of, C-21
- partitioned spatial indexes, 4-3
 - exchanging partitions, 4-5
- PERCENTAGE_TO_MEASURE function, 16-61
- performance and tuning information, 1-16
 - for Spatial operators, 1-13
- PL/SQL and SQL examples, 1-17
- point
 - data, 1-5
 - illustration and examples of point-only geometry, 2-20
 - locating on geometric segment, 7-12
 - LRS, 7-5
 - on surface of polygon, 15-31
 - oriented point, 2-21
 - projection of onto geometric segment, 7-5, 7-13
 - shape, 7-2
- point of interest (POI)
 - GC_POI_<suffix> table, 5-14
- POINT_AT_BEARING function, 20-22
- polygon
 - area of, 15-8
 - boundary of, 1-11

- buffer, 15-10
- centroid, 15-13
- compound, 2-10
- exterior and interior rings, 2-8, 2-15, 2-17
- point on surface, 15-31
- self-crossing not supported, 1-4
- self-intersecting polygons, 15-39
 - with hole (LRS measure values), 7-5
- polygon collection, 2-17
- polygon data, 1-5
- POLYGONTOLINE function, 20-24
- populating
 - measure, 7-3
- pre_geocoded_location element, C-12
- pre_geocoded_locations attribute
 - of route request, C-11
- PREPARE_FOR_TTS procedure, 20-25
- primary filter, 1-8, 4-8, 4-9
- primitive types, 1-3
- problems in current release, 6-40
 - geodetic data, 6-5
- PROJECT_PT function, 16-63
- projected coordinates, 1-6, 6-2
- projections, 7-5, 7-13
 - MDSYS.SDO_PROJECTIONS_OLD_FORMAT table, 6-33
 - MDSYS.SDO_PROJECTIONS_OLD_SNAPSHOT table, 6-33
 - PROJECT_PT function, 16-63

Q

- quadtree indexes
 - deprecated feature of Spatial, 1-9
- quality
 - degradation of R-tree index, 19-10
 - R-tree, 1-10
- QUALITY_DEGRADATION function, 19-10
- query, 4-7
- query model for Spatial, 1-8
- query window, 4-7
- querying geometric segments
 - subprograms for, 16-2

R

- range
 - measure, 7-5
- README file
 - for Spatial, GeoRaster, and topology and network data models, 1-18
- rebuilding
 - spatial indexes, 10-4
- rectangle
 - minimum bounding, 19-7
 - type, 2-10
- rectification of geometries, 20-26
- RECTIFY_GEOMETRY function, 20-26
- REDEFINE_GEOM_SEGMENT procedure, 16-65
- redefining

- geometric segment, 7-9
- RELATE function, 15-3
 - See also* SDO_RELATE operator
- release number (Spatial)
 - retrieving, 1-16
- REMOVE_DUPLICATE_VERTICES function, 20-27
- replication
 - multimaster, B-3
 - object, B-2
- RESET_MEASURE procedure, 16-67
- restrictions in current release, 6-40
 - geodetic data, 6-5
- return_detailed_geometries attribute
 - of route request, C-11
- return_driving_directions attribute
 - of route request, C-11
- return_hierarchical_driving_directions attribute
 - of route request, C-11
- return_route_geometry attribute
 - of batch route request, C-17
 - of route request, C-11
- return_segment_geometry attribute
 - of batch route request, C-17
 - of route request, C-11
- reverse geocoding, 14-10
- REVERSE_GEOCODE function, 14-10
- REVERSE_GEOMETRY function, 16-69
- REVERSE_LINestring function, 20-29
- REVERSE_MEASURE function, 16-71
- REVOKE_PREFERENCE_FOR_OP procedure, 13-31
- ring
 - exterior and interior polygon, 2-8
 - extracting from a geometry, 20-11
- road segments
 - GC_ROAD_SEGMENT_<suffix> table, 5-18
- road_preference attribute
 - of batch route request, C-17
 - of route request, C-11
- roads
 - GC_ROAD_<suffix> table, 5-16
- rollback segment
 - R-tree index creation, 4-1
- route geometry
 - in route request, C-11
- route requests, C-10
 - DTD, C-9
 - example, C-5
 - previously geocoded locations, C-8
 - input_location element, C-12
 - pre_geocoded_location element, C-12
- route responses
 - DTD, C-12
 - example, C-5
 - previously geocoded locations, C-9
- route_preference attribute
 - of route request, C-11
- route_request element, C-10
- routing engine
 - configuring, C-2
 - data structures used by, C-20

- deploying, C-2
- overview, C-1
- XML API, C-3
- R-tree indexes
 - description of indexing process, 1-9
 - quality degradation, 19-10
 - rebuilding, 10-4
 - requirements before creating, 4-1
 - sequence object, 2-35
 - when to use, 1-9
- R-tree quality, 1-10

S

- scaling
 - geometric segment, 7-11
- schemas
 - creating index on table in another schema, 4-3
 - invoking operators on table in another schema, 4-12
- SDO_ADDR_ARRAY data type, 5-7
- SDO_AGGR_CENTROID aggregate function, 12-2
- SDO_AGGR_CONCAT_LINES aggregate function, 12-3
- SDO_AGGR_CONVEXHULL aggregate function, 12-5
- SDO_AGGR_LRS_CONCAT aggregate function, 12-6
- SDO_AGGR_MBR aggregate function, 12-8
- SDO_AGGR_UNION aggregate function, 12-9
 - complex examples, D-5
- SDO_ANGLE_UNITS table, 6-29
- SDO_ANYINTERACT operator, 11-3
- SDO_ARC_DENSIFY function, 15-6
- SDO_AREA function, 15-8
- SDO_AREA_UNITS view, 6-29
- SDO_BUFFER function, 15-10
- SDO_CENTROID function, 15-13
- SDO_CONTAINS operator, 11-5
- SDO_CONVEXHULL function, 15-15
- SDO_COORD_AXES table, 6-8
- SDO_COORD_AXIS_NAMES table, 6-9
- SDO_COORD_OP_METHODS table, 6-9
- SDO_COORD_OP_PARAM_USE table, 6-10
- SDO_COORD_OP_PARAM_VALS table, 6-10
- SDO_COORD_OP_PARAMS table, 6-11
- SDO_COORD_OP_PATHS table, 6-11
- SDO_COORD_OPS table, 6-12
- SDO_COORD_REF_SYS table, 6-13
- SDO_COORD_REF_SYSTEM view, 6-15
- SDO_COORD_SYS table, 6-15
- SDO_COVEREDBY operator, 11-6
- SDO_COVERS operator, 11-7
- SDO_CRS_COMPOUND view, 6-15
- SDO_CRS_ENGINEERING view, 6-16
- SDO_CRS_GEOCENTRIC view, 6-16
- SDO_CRS_GEOGRAPHIC2D view, 6-17
- SDO_CRS_GEOGRAPHIC3D view, 6-17
- SDO_CRS_PROJECTED view, 6-18
- SDO_CRS_VERTICAL view, 6-18

SDO_CS package, 13-1
 ADD_PREFERENCE_FOR_OP, 13-4
 CONVERT_NADCON_TO_XML, 13-6
 CONVERT_NTV2_TO_XML, 13-8
 CONVERT_XML_TO_NADCON, 13-10
 CONVERT_XML_TO_NTV2, 13-12
 CREATE_CONCATENATED_OP, 13-14
 CREATE_OBVIOUS_EPSG_RULES, 13-15
 CREATE_PREF_CONCATENATED_OP, 13-16
 DELETE_ALL_EPSG_RULES, 13-18
 DELETE_OP, 13-19
 DETERMINE_CHAIN, 13-20
 DETERMINE_DEFAULT_CHAIN, 13-22
 FIND_GEOG_CRS, 13-23
 FIND_PROJ_CRS, 13-25
 FROM_OGC_SIMPLEFEATURE_SRS, 13-27
 FROM_USNG, 13-28
 MAP_EPSG_SRID_TO_ORACLE, 13-29
 MAP_ORACLE_SRID_TO_EPSG, 13-30
 REVOKE_PREFERENCE_FOR_OP, 13-31
 TO_OGC_SIMPLEFEATURE_SRS, 13-32
 TO_USNG, 13-33
 TRANSFORM, 13-35
 TRANSFORM_LAYER, 13-38
 UPDATE_WKTS_FOR_ALL_EPSG_CRS, 13-40
 UPDATE_WKTS_FOR_EPSG_CRS, 13-41
 UPDATE_WKTS_FOR_EPSG_DATUM, 13-42
 UPDATE_WKTS_FOR_EPSG_ELLIPS, 13-43
 UPDATE_WKTS_FOR_EPSG_OP, 13-44
 UPDATE_WKTS_FOR_EPSG_PARAM, 13-45
 UPDATE_WKTS_FOR_EPSG_PM, 13-46
 VALIDATE_WKT, 13-47
 VIEWPORT_TRANSFORM, 13-48
 SDO_DATUM_ENGINEERING view, 6-19
 SDO_DATUM_GEODETIC view, 6-20
 SDO_DATUM_VERTICAL view, 6-20
 SDO_DATUMS table, 6-21
 SDO_DATUMS_OLD_FORMAT table, 6-30
 SDO_DATUMS_OLD_SNAPSHOT table, 6-30
 SDO_DIFFERENCE function, 15-17
 SDO_DIST_UNITS view, 6-32
 SDO_DISTANCE function, 15-19
 SDO_ELEM_INFO attribute, 2-7
 SDO_ELEM_INFO_ARRAY type, 2-5
 SDO_ELLIPSOIDS table, 6-22
 SDO_ELLIPSOIDS_OLD_FORMAT table, 6-32
 SDO_ELLIPSOIDS_OLD_SNAPSHOT table, 6-32
 SDO_EQUAL operator, 11-8
 SDO_ETYPE value, 2-8, 2-9
 SDO_FILTER operator, 11-9
 SDO_GCDR package, 14-1
 GEOCODE, 14-2
 GEOCODE_ADDR, 14-3
 GEOCODE_ADDR_ALL, 14-5
 GEOCODE_ALL, 14-7
 GEOCODE_AS_GEOMETRY, 14-9
 REVERSE_GEOCODE, 14-10
 SDO_GEO_ADDR data type and constructors, 5-4
 SDO_GEOM package
 RELATE, 15-3
 SDO_ARC_DENSIFY, 15-6
 SDO_AREA, 15-8
 SDO_BUFFER, 15-10
 SDO_CENTROID, 15-13
 SDO_CONVEXHULL, 15-15
 SDO_DIFFERENCE, 15-17
 SDO_DISTANCE, 15-19
 SDO_INTERSECTION, 15-21
 SDO_LENGTH, 15-23
 SDO_MAX_MBR_ORDINATE, 15-25
 SDO_MBR, 15-27
 SDO_MIN_MBR_ORDINATE, 15-29
 SDO_POINTONSURFACE, 15-31
 SDO_UNION, 15-33
 SDO_XOR, 15-35
 VALIDATE_GEOMETRY_WITH_CONTEXT, 15-37
 VALIDATE_LAYER_WITH_CONTEXT, 15-41
 WITHIN_DISTANCE, 15-43
 SDO_GEOMETRY object type, 2-5
 constructors, 2-13
 embedding in user-defined type, 9-1, 9-4
 in function-based indexes, 9-3
 methods (member functions), 2-11
 SDO_GTYPE attribute, 2-5
 constraining data to a geometry type, 4-2
 Get_Dims method, 2-11
 Get_GType method, 2-11
 Get_LRS_Dim method, 2-11
 SDO_INDEX_TABLE entry in index metadata
 views, 2-34
 SDO_INDX_DIMS keyword, 7-7, 10-2
 SDO_INSIDE operator, 11-12
 SDO_INTERPRETATION value, 2-8
 SDO_INTERSECTION function, 15-21
 SDO_JOIN operator, 11-13
 SDO_KEYWORDARRAY data type, 5-7
 SDO_LENGTH function, 15-23
 SDO_LRS package
 CLIP_GEOM_SEGMENT, 16-5
 CONCATENATE_GEOM_SEGMENTS, 16-7
 CONNECTED_GEOM_SEGMENTS, 16-10
 CONVERT_TO_LRS_DIM_ARRAY, 16-12
 CONVERT_TO_LRS_GEOM, 16-14
 CONVERT_TO_LRS_LAYER, 16-16
 CONVERT_TO_STD_DIM_ARRAY, 16-18
 CONVERT_TO_STD_GEOM, 16-19
 CONVERT_TO_STD_LAYER, 16-20
 DEFINE_GEOM_SEGMENT, 16-22
 DYNAMIC_SEGMENT, 16-25
 FIND_LRS_DIM_POS, 16-27
 FIND_MEASURE, 16-28
 FIND_OFFSET, 16-30
 GEOM_SEGMENT_END_MEASURE, 16-32
 GEOM_SEGMENT_END_PT, 16-33
 GEOM_SEGMENT_LENGTH, 16-34
 GEOM_SEGMENT_START_MEASURE, 16-35
 GEOM_SEGMENT_START_PT, 16-36
 GET_MEASURE, 16-37
 GET_NEXT_SHAPE_PT, 16-38

GET_NEXT_SHAPE_PT_MEASURE, 16-40
 GET_PREV_SHAPE_PT, 16-42
 GET_PREV_SHAPE_PT_MEASURE, 16-44
 IS_GEOM_SEGMENT_DEFINED, 16-46
 IS_MEASURE_DECREASING, 16-47
 IS_MEASURE_INCREASING, 16-48
 IS_SHAPE_PT_MEASURE, 16-49
 LOCATE_PT, 16-51
 LRS_INTERSECTION, 16-53
 MEASURE_RANGE, 16-55
 MEASURE_TO_PERCENTAGE, 16-56
 OFFSET_GEOM_SEGMENT, 16-58
 PERCENTAGE_TO_MEASURE, 16-61
 PROJECT_PT, 16-63
 REDEFINE_GEOM_SEGMENT, 16-65
 RESET_MEASURE, 16-67
 REVERSE_GEOMETRY, 16-69
 REVERSE_MEASURE, 16-71
 SET_PT_MEASURE, 16-73
 SPLIT_GEOM_SEGMENT, 16-76
 TRANSLATE_MEASURE, 16-78
 VALID_GEOM_SEGMENT, 16-80
 VALID_LRS_PT, 16-81
 VALID_MEASURE, 16-82
 VALIDATE_LRS_GEOMETRY, 16-84
 SDO_MAX_MBR_ORDINATE function, 15-25
 SDO_MBR function, 15-27
 SDO_MIGRATE package
 TO_CURRENT, 17-2
 SDO_MIN_MBR_ORDINATE function, 15-29
 SDO_NN operator, 11-16
 complex examples, D-3
 optimizer hints, 11-18
 SDO_NN_DISTANCE ancillary operator, 11-20
 SDO_NN_DISTANCE operator
 complex examples, D-3
 SDO_ON operator, 11-22
 SDO_ORDINATE_ARRAY type, 2-5
 SDO_ORDINATES attribute, 2-10
 SDO_OVERLAPBDYDISJOINT operator, 11-23
 SDO_OVERLAPBDYINTERSECT operator, 11-25
 SDO_OVERLAPS operator, 11-27
 SDO_POINT attribute, 2-7
 SDO_POINT_TYPE object type, 2-5
 SDO_POINTONSURFACE function, 15-31
 SDO_PREFERRED_OPS_SYSTEM table, 6-23
 SDO_PREFERRED_OPS_USER table, 6-23
 SDO_PRIME_MERIDIANS table, 6-24
 SDO_PROJECTIONS_OLD_FORMAT table, 6-33
 SDO_PROJECTIONS_OLD_SNAPSHOT table, 6-33
 SDO_REGAGGR object type, 18-5, 18-18
 SDO_REGAGGRSET object type, 18-5, 18-18
 SDO_REGION object type, 18-21
 SDO_REGIONSET object type, 18-21
 SDO_RELATE operator, 11-29
 SDO_ROWIDPAIR object type, 11-13
 SDO_ROWIDSET data type, 11-13
 SDO_RTREE_SEQ_NAME sequence object, 2-35
 SDO_SAM package, 18-1
 AGGREGATES_FOR_GEOMETRY, 18-2
 AGGREGATES_FOR_LAYER, 18-4
 BIN_GEOMETRY, 18-6
 BIN_LAYER, 18-8
 COLOCATED_REFERENCE_FEATURES, 18-10
 SIMPLIFY_GEOMETRY, 18-12
 SIMPLIFY_LAYER, 18-14
 SPATIAL_CLUSTERS, 18-16
 TILED_AGGREGATES, 18-17
 TILED_BINS, 18-20
 SDO_SRID attribute, 2-7
 SDO_STARTING_OFFSET value, 2-7
 SDO_TFM_CHAIN type, 6-7
 SDO_TOUCH operator, 11-33
 SDO_TUNE package
 AVERAGE_MBR, 19-2
 ESTIMATE_RTREE_INDEX_SIZE, 19-4
 EXTENT_OF, 19-7
 MIX_INFO, 19-8
 QUALITY_DEGRADATION, 19-10
 SDO_UNION function, 15-33
 SDO_UNIT column
 in SDO_AREA_UNITS table, 6-30
 in SDO_DIST_UNITS table, 6-32
 SDO_UNITS_OF_MEASURE table, 6-24
 SDO_UTIL package
 APPEND, 20-3
 CIRCLE_POLYGON, 20-4
 CONCAT_LINES, 20-6
 CONVERT_UNIT, 20-8
 ELLIPSE_POLYGON, 20-9
 EXTRACT, 20-11
 FROM_WKBGEOMETRY, 20-13
 FROM_WKTGEOMETRY, 20-15
 GETNUMELEM, 20-17
 GETNUMVERTICES, 20-18
 GETVERTICES, 20-19
 INITIALIZE_INDEXES_FOR_TTS, 20-21
 POINT_AT_BEARING, 20-22
 POLYGONTOLINE, 20-24
 PREPARE_FOR_TTS, 20-25
 RECTIFY_GEOMETRY, 20-26
 REMOVE_DUPLICATE_VERTICES, 20-27
 REVERSE_LINestring, 20-29
 SIMPLIFY, 20-30
 TO_GMLGEOMETRY, 20-33
 TO_WKBGEOMETRY, 20-39
 TO_WKTGEOMETRY, 20-41
 VALIDATE_WKBGEOMETRY, 20-43
 VALIDATE_WKTGEOMETRY, 20-45
 SDO_VERSION function, 1-16
 SDO_WITHIN_DISTANCE operator, 11-35
 complex examples, D-2
 SDO_XOR function, 15-35
 SDOAGGRTYPE object type, 1-14
 secondary filter, 1-8, 4-9
 segment geometry
 in route request, C-11
 segments
 geometric, 7-2
 self-crossing line strings and polygons, 1-4

- self-intersecting polygons, 15-39
- semi-major axis, 6-22, 6-33
- semi-minor axis, 6-22
- sequence object for R-tree index, 2-35
- SET_PT_MEASURE procedure, 16-73
- shape point, 7-2
 - determining if measure value is a shape point, 16-49
 - getting measure of next, 16-40
 - getting measure of previous, 16-44
 - getting next, 16-38
 - getting previous, 16-42
- SIGN_POST table
 - routing engine use of, C-21
- simple element, 2-8
- simple features (OGC)
 - Oracle Spatial conformance, 1-16
- simplification of geometries, 20-30
- SIMPLIFY function, 20-30
- SIMPLIFY_GEOMETRY function, 18-12
- SIMPLIFY_LAYER procedure, 18-14
- size requirements (hardware) for spatial applications, 1-17
- SORT_AREA_SIZE parameter
 - R-tree index creation, 4-2
- sort_by_distance attribute
 - of batch route request, C-17
- spatial aggregate functions
 - See aggregate functions
- spatial analysis and mining
 - conceptual and usage information, 8-1
 - function reference information, 18-1
- spatial binning, 8-3
 - See also bins
- spatial clustering, 8-4
- spatial correlation, 8-3
- spatial data mining
 - conceptual and usage information, 8-1
 - function reference information, 18-1
- spatial data structures, 2-1
- spatial data types, 2-1
- spatial index
 - See index
- spatial join, 4-11
 - SDO_JOIN operator, 11-13
- spatial operators
 - See operators
- spatial query, 4-7
- spatial reference systems
 - conceptual and usage information, 6-1
 - example, 6-41
 - subprogram reference information, 13-1
- spatial routing engine
 - See routing engine
- SPATIAL_CLUSTERS function, 18-16
- spheroids
 - MDSYS.SDO_ELLIPSOIDS_OLD_SNAPSHOT table, 6-32
- SPLIT_GEOM_SEGMENT procedure, 16-76
- splitting

- geometric segment, 7-10
- SQL and PL/SQL examples, 1-17
- SQL Multimedia
 - FROM_WKBGEOMETRY function, 20-13
 - FROM_WKTGEOMETRY function, 20-15
 - TO_WKBGEOMETRY function, 20-39
 - TO_WKTGEOMETRY function, 20-41
 - VALIDATE_WKBGEOMETRY function, 20-43
 - VALIDATE_WKTGEOMETRY function, 20-45
- SQL Multimedia standard
 - ST_CoordDim method, 2-12
 - ST_IsValid method, 2-12
- SQL statements
 - for indexing spatial data, 10-1
- SQL*Loader, 3-1
- SRID
 - 0 (zero) special case with SDO_CS.VIEWPORT_TRANSFORM function, 13-48
 - in USER_SDO_GEOM_METADATA, 2-31
 - mapping Oracle to EPSG, 13-30
 - SDO_SRID attribute in SDO_GEOMETRY, 2-7
- ST_CoordDim method, 2-12
- ST_IsValid method, 2-12
- start location for route, C-10
- subprograms
 - coordinate system transformation, 13-1
 - data mining, 18-1
 - geocoding, 14-1
 - geometry, 15-1
 - linear referencing, 16-1
 - SDO_MIGRATE, 17-1
 - spatial analysis, 18-1
 - tuning, 19-1
 - utility, 20-1

T

- table names
 - restrictions on spatial table names, 2-30
- TABLE_NAME (in USER_SDO_GEOM_METADATA), 2-30
- TFM_PLAN object type, 6-7
- three-dimensional (3D)
 - formats of LRS functions, 7-7
 - not supported with geodetic data, 6-40
- TILED_AGGREGATES function, 18-17
- TILED_BINS function, 18-20
- time_unit attribute
 - of route request, C-11
- TO_81X procedure
 - use TO_CURRENT instead, 17-1
- TO_CURRENT procedure and function, 17-2
- TO_GMLGEOMETRY function, 20-33
- TO_OGC_SIMPLEFEATURE_SRS function, 13-32
- TO_USNG function, 13-33
- TO_WKBGEOMETRY function, 20-39
- TO_WKTGEOMETRY function, 20-41
- tolerance, 1-6
 - with LRS functions, 7-15
- TOUCH

- SDO_TOUCH operator, 11-33
 - topological relationship, 1-12
- transactional insertion of spatial data, 3-3
- TRANSFORM function, 13-35
- TRANSFORM_LAYER procedure, 13-38
 - table for transformed layer, 13-39
- transformation, 6-2
- TRANSLATE_MEASURE function, 16-78
- transportable tablespaces
 - initializing spatial indexes, 20-21
 - preparing for when using spatial indexes, 20-25
- tuning and performance information, 1-16
 - for spatial operators, 1-13
- tuning subprograms, 19-1
- two-tier query model, 1-8
- type zero (0) element, 2-23

U

- unformatted addresses, 5-2
- union, 15-33
- unit of measurement
 - MDSYS tables, 2-35
- UNIT_NAME column
 - in SDO_ANGLE_UNITS table, 6-29
 - in SDO_AREA_UNITS table, 6-30
 - in SDO_DIST_UNITS table, 6-32
- unknown CRS coordinate reference system, 6-40
- UPDATE_WKTS_FOR_ALL_EPSG_CRS
 - procedure, 13-40
- UPDATE_WKTS_FOR_EPSG_CRS procedure, 13-41
- UPDATE_WKTS_FOR_EPSG_DATUM
 - procedure, 13-42
- UPDATE_WKTS_FOR_EPSG_ELLIPS
 - procedure, 13-43
- UPDATE_WKTS_FOR_EPSG_OP procedure, 13-44
- UPDATE_WKTS_FOR_EPSG_PARAM
 - procedure, 13-45
- UPDATE_WKTS_FOR_EPSG_PM procedure, 13-46
- upgrading
 - data to current Spatial release, 17-2
 - GeoRaster, A-1
 - LRS data, A-1
 - Spatial to current Oracle Database release, A-1
- U.S. National Grid
 - SDO_CS.FROM_USNG function, 13-28
 - SDO_CS.TO_USNG function, 13-33
 - support in Oracle Spatial, 6-41
- use cases
 - for coordinate system transformation, 6-7
- USER_SDO_GEOM_METADATA view, 2-29
- USER_SDO_INDEX_INFO view, 2-31
- USER_SDO_INDEX_METADATA view, 2-32
- user-defined coordinate reference system, 6-34
- user-defined data types
 - embedding SDO_GEOMETRY objects in, 9-1, 9-4
- UTF8 character set
 - NLS_LENGTH_SEMANTICS setting for spatial queries, 4-6
- utility subprograms, 20-1

V

- VALID_GEOM_SEGMENT function, 16-80
- VALID_LRS_PT function, 16-81
- VALID_MEASURE function, 16-82
- VALIDATE_GEOMETRY_WITH_CONTEXT
 - function, 15-37
- VALIDATE_LAYER_WITH_CONTEXT
 - procedure, 15-41
- VALIDATE_LRS_GEOMETRY function, 16-84
- VALIDATE_WKBGEOMETRY function, 20-43
- VALIDATE_WKT function, 13-47
- VALIDATE_WKTGEOMETRY function, 20-45
- vendor attribute
 - of route request, C-11
- version number (Spatial)
 - retrieving, 1-16
- VERTEX_SET_TYPE data type, 20-19
- VERTEX_TYPE object type, 20-19
- vertices
 - maximum number in SDO_GEOMETRY object, 2-5
 - removing duplicate, 20-27
 - returning geometry coordinates as, 20-19
- VIEWPORT_TRANSFORM function, 13-48
- views
 - ALL_SDO_GEOM_METADATA, 2-29
 - ALL_SDO_INDEX_INFO, 2-32
 - ALL_SDO_INDEX_METADATA, 2-32
 - USER_SDO_GEOM_METADATA, 2-29
 - USER_SDO_INDEX_INFO, 2-31
 - USER_SDO_INDEX_METADATA, 2-32

W

- well-known binary (WKB)
 - See WKB
- well-known text (WKT)
 - See WKT
- WITHIN_DISTANCE function, 15-43
 - See also SDO_WITHIN_DISTANCE operator
- WKB
 - FROM_WKBGEOMETRY function, 20-13
 - Get_WKB method, 2-11
 - TO_WKBGEOMETRY function, 20-39
 - VALIDATE_WKBGEOMETRY function, 20-43
- WKT, 6-26
 - FROM_WKTGEOMETRY function, 20-15
 - Get_WKT method, 2-12
 - procedures for updating, 6-28
 - TO_WKTGEOMETRY function, 20-41
 - UPDATE_WKTS_FOR_ALL_EPSG_CRS
 - procedure, 13-40
 - UPDATE_WKTS_FOR_EPSG_CRS
 - procedure, 13-41
 - UPDATE_WKTS_FOR_EPSG_DATUM
 - procedure, 13-42
 - UPDATE_WKTS_FOR_EPSG_ELLIPS
 - procedure, 13-43
 - UPDATE_WKTS_FOR_EPSG_OP
 - procedure, 13-44

UPDATE_WKTS_FOR_EPSG_PARAM
 procedure, 13-45
UPDATE_WKTS_FOR_EPSG_PM
 procedure, 13-46
VALIDATE_WKTGEOMETRY function, 20-45
 validating (given SRID), 13-47
WKTEXT column of MDSYS.CS_SRS table, 6-26
 procedures for updating value, 6-28

X

XML API
 routing engine, C-3
XOR
 SDO_XOR function, 15-35

Z

zero
 SRID value used with SDO_CS.VIEWPORT_
 TRANSFORM function, 13-48
 type 0 element, 2-23

