Oracle® Spatial

User's Guide and Reference

10*g* Release 1 (10.1)

Part No. B10826-01

December 2003

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

ORACLE®

Oracle Spatial User's Guide and Reference, 10*g* Release 1 (10.1)

Part No. B10826-01

Primary Author: Chuck Murray

Contributors: Dan Abugov, Nicole Alexander, Bruce Blackwell, Janet Blowney, Dan Geringer, Albert Godfrind, Ravi Kothuri, Richard Pitts, Siva Ravada, Jack Wang, Jeffrey Xie

Contents

Part I Conceptual and Usage Information

1 Spatial Concepts

7 Linear Referencing System

8 Spatial Analysis and Mining

9 Extending Spatial Indexing Capabilities

Part II Reference Information

10 SQL Statements for Indexing Spatial Data

11 SDO_GEOMETRY Object Type Methods

12 Spatial Operators

13 Geometry Subprograms

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Spatial User's Guide and Reference, 10*g* Release 1 (10.1)

Part No. B10826-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825 Attn: Spatial Documentation
- Postal service:
  Oracle Corporation
  Oracle Spatial Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  USA

If you would like a reply, please give your name and contact information.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

The *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 10*g*. 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 10*g*. 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 10*g* 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 10*g* table partitioning, you must have the Enterprise Edition and the Partitioning Option.

For information about the differences between Oracle Database 10*g* Standard Edition and Oracle Database 10*g* 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.

This preface contains these topics:

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. 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 additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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.

Organization

This guide has two main parts (conceptual and usage information, and reference information) and a third part with supplementary information. The first part 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).

This guide has the following elements.

Part I, "Conceptual and Usage Information"

Contains chapters with conceptual and usage information.

Chapter 1, "Spatial Concepts"

Explains important concepts and techniques that you need to know to use Spatial.

Chapter 2, "Spatial Data Types and Metadata"

Explains the data types and metadata for Spatial. It includes a complete simplified example of using Spatial, as well as several examples of spatial geometries.

Chapter 3, "Loading Spatial Data"

Explains how to load spatial data.

Chapter 4, "Indexing and Querying Spatial Data"

Explains how to index and query spatial data.

Chapter 5, "Geocoding Address Data"

Provides conceptual and usage information about support for geocoding.

Chapter 6, "Coordinate Systems (Spatial Reference Systems)"

Provides conceptual and usage information about coordinate system (spatial reference system) support.

Chapter 7, "Linear Referencing System"

Provides conceptual and usage information about the Oracle Spatial linear referencing system (LRS).

Chapter 8, "Spatial Analysis and Mining"

Provides conceptual and usage information about the Oracle Spatial analysis and mining features for data mining applications.

Chapter 9, "Extending Spatial Indexing Capabilities"

Explains how to extend the capabilities of Oracle Spatial indexing.

Part II, "Reference Information"

Contains chapters with reference information.

Chapter 10, "SQL Statements for Indexing Spatial Data"

Provides the syntax and semantics for SQL indexing statements.

Chapter 11, "SDO_GEOMETRY Object Type Methods"

Provides the syntax and semantics for methods used with the spatial object data type.

Chapter 12, "Spatial Operators"

Provides the syntax and semantics for operators used with the spatial object data type.

Chapter 13, "Geometry Subprograms"

Provides the syntax and semantics for the geometric functions and procedures.

Chapter 14, "Spatial Aggregate Functions"

Provides the syntax and semantics for the spatial aggregate functions.

Chapter 15, "Coordinate System Transformation Subprograms"

Provides the syntax and semantics for the coordinate system transformation functions and procedures.

Chapter 16, "Linear Referencing Subprograms"

Provides the syntax and semantics for the functions and procedures related to the Oracle Spatial linear referencing system (LRS).

Chapter 17, "SDO_MIGRATE Procedure"

Provides the syntax and semantics for the migration procedure.

Chapter 18, "Spatial Tuning Subprograms"

Provides the syntax and semantics for the spatial tuning functions and procedures.

Chapter 19, "Spatial Utility Subprograms"

Provides the syntax and semantics for the spatial utility functions and procedures.

Chapter 20, "Geocoding Subprograms"

Provides the syntax and semantics for the geocoding functions and procedures.

Chapter 21, "Spatial Analysis and Mining Subprograms"

Provides the syntax and semantics for the spatial analysis and mining functions and procedures.

Part III, "Supplementary Information"

Contains appendixes and a glossary.

Appendix A, "Installation, Compatibility, and Upgrade"

Describes installation, compatibility, and upgrade issues.

Appendix B, "Oracle Locator"

Describes Oracle Locator.

Appendix C, "Complex Spatial Queries: Examples"

Provides examples, with explanations, of queries that are more complex than the examples in the reference chapters.

Glossary

Defines important terms.

Technologies Released Separately

Technologies of interest to spatial application developers, but not officially part of Oracle Spatial, are sometimes made available through the Oracle Technology Network (OTN). To access the Spatial page on OTN, go to

`http://otn.oracle.com/products/spatial/`

Related Documentation

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://otn.oracle.com/membership`

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

`http://otn.oracle.com/documentation`

Conventions

The following conventions are used in this guide:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface text indicates a term defined in the text, the glossary, or in both locations.
`monospace text`	Monospace text is used for the names of parameters, files, and directory paths. It is also used for SQL and PL/SQL code examples.
italic text	Italic text is used for book titles, emphasis, and some special terms.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

New and Changed Features

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

GeoRaster

GeoRaster is a feature of Oracle Spatial that lets you store, index, query, analyze, and deliver georaster data, that is, raster image data and its associated Spatial vector geometry data, plus metadata. GeoRaster provides Oracle Spatial data types and an object-relational schema for storing multidimensional grid layers and digital images that can be referenced to positions on the Earth's surface or a local coordinate system.

Information about GeoRaster is in a separate manual: *Oracle Spatial GeoRaster*.

Topology and Network Management

The topology and network management capabilities of Oracle Spatial let you work with data about nodes, edges, and faces in a topology, and nodes and edges in a network. For example, United States Census geographic data is provided in terms of nodes, chains (line strings), and polygons (faces). You can store information about the topological elements and feature layers in Oracle Spatial tables and metadata views, and then you can perform certain Spatial operations referencing the topological elements, for example, finding which chains (such as streets) have any spatial interaction with a specific polygon entity (such as a park).

Information about topology and network management is in a separate manual: *Oracle Spatial Topology and Network Data Models*.

Spatial Analysis and Mining

You can use new spatial analysis and mining subprograms in Oracle Data Mining (ODM) applications. See Chapter 8 for conceptual and usage information, and Chapter 21 for reference information about each subprogram.

Geocoding

The geocoding capabilities of Oracle Spatial let you geocode unformatted addresses. See Chapter 5 for conceptual and usage information, and Chapter 20 for reference information about each subprogram.

Quadtree Indexing Discouraged; R-Tree Indexing Encouraged

The use of spatial quadtree indexes is discouraged. You are strongly encouraged to use R-tree indexing for spatial indexes, unless you need to continue using quadtree indexes for special situations. Significant performance improvements have been made to spatial R-tree indexing for this release.

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.

For information about spatial indexing, see Section 1.7 and Chapter 4.

New Utility Subprograms

The SDO_UTIL package contains the following new subprograms:

- SDO_UTIL.APPEND appends one geometry to another geometry to create a new geometry
- 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.CIRCLE_POLYGON returns the polygon geometry that approximates and is covered by a specified circle.
- SDO_UTIL.ELLIPSE_POLYGON returns the polygon geometry that approximates and is covered by a specified ellipse.

- 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.REMOVE_DUPLICATE_VERTICES removes duplicate (redundant) vertices from a geometry.
- SDO_UTIL.REVERSE_LINESTRING returns a line string geometry with the vertices of the input geometry in reverse order.
- SDO_UTIL.SIMPLIFY simplifies the input geometry, based on a threshold value, using the Douglas-Peucker algorithm.
- SDO_UTIL.TO_GMLGEOMETRY 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.

See Chapter 19 for reference information about the utility subprograms.

New Operators

SDO_JOIN performs a spatial join based on one or more topological relations. (SDO_JOIN is technically not an operator, but a table function; 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.)

The following new operators are convenient alternatives to using SDO_RELATE with a specified `mask` value: SDO_ANYINTERACT, SDO_CONTAINS, SDO_COVEREDBY, SDO_COVERS, SDO_EQUAL, SDO_INSIDE, SDO_ON, SDO_OVERLAPBDYDISJOINT, SDO_OVERLAPBDYINTERSECT, SDO_OVERLAPS (OVERLAPBDYDISJOINT or OVERLAPBDYINTERSECT), and SDO_TOUCH.

See Chapter 12 for reference information about these operators.

SDO_NN Operator Behavior Changes

If you do not specify the `param` parameter with the SDO_NN operator, the operator returns all rows in increasing distance order from `geometry2`.

In the `param` parameter, you can specify `sdo_batch_size=0`, which causes Spatial to calculate a batch size that is suitable for the result set size; however, there are performance considerations, as explained in the Usage Notes for the SDO_NN operator in Chapter 12. (This feature worked in the previous release, but was not documented.)

New Spatial Aggregate Function

SDO_AGGR_CONCAT_LINES returns a geometry that concatenates the specified line or multiline geometries.

See Chapter 14 for reference information about spatial aggregate functions.

New Coordinate Systems Function: VALIDATE_WKT

The new SDO_CS.VALIDATE_WKT function validates the well-known text (WKT) description associated with a specified SRID.

See Chapter 6 for conceptual and usage information about coordinate systems, and Chapter 15 for reference information about coordinate system transformation subprograms.

MBRs Supported with Geodetic Data

Minimum bounding rectangles (MBRs) can now be geodetic or non-geodetic. In previous releases, MBRs needed to be non-geodetic.

The following functions, which were not supported with geodetic data in the previous release, are now supported for use with geodetic data:

- SDO_GEOM.SDO_MBR, SDO_GEOM.SDO_MIN_MBR_ORDINATE, and SDO_GEOM.SDO_MAX_MBR_ORDINATE (described in Chapter 13)
- SDO_AGGR_MBR (described in Chapter 14)

Because geodetic MBRs are easier to use than the previous technique using the SDO_CS.VIEWPORT_TRANSFORM function (described in Chapter 15), that function is deprecated, and it will not be supported in future releases of Spatial.

New and Changed LRS Subprograms

The new SDO_LRS.FIND_OFFSET function returns the signed offset (shortest distance) between a point and a geometric segment.

The SDO_LRS.PROJECT_PT function includes a new optional output parameter `offset`, which stores the signed offset of the projected point from the geometric segment.

See Chapter 16 for reference information about linear referencing system (LRS) subprograms.

Tolerance with LRS Subprograms

In the current version of this guide, `tolerance` is shown as a required parameter for many LRS subprograms where it had previously been shown as optional. Applications that used `tolerance` as an optional parameter in such cases will continue to work in this release; however, such usage is deprecated and will not be supported in future releases. If you use an LRS subprogram format in which `tolerance` is shown as required, you should specify that parameter.

See also Section 7.6, which contains new information about tolerance values with LRS subprograms.

New Tuning Function: ESTIMATE_RTREE_INDEX_SIZE

The new SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE function, described in Chapter 18, estimates the maximum number of megabytes needed for an R-tree spatial index table.

Deprecated Tuning Subprograms

The following SDO_TUNE subprograms specific to spatial quadtree indexing are deprecated:

- SDO_TUNE.ESTIMATE_INDEX_PERFORMANCE
- SDO_TUNE.ESTIMATE_TILING_LEVEL
- SDO_TUNE.ESTIMATE_TILING_TIME
- SDO_TUNE.ESTIMATE_TOTAL_NUMTILES
- SDO_TUNE.HISTOGRAM_ANALYSIS

Information about these subprograms 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.

New GML Support Function: TO_GMLGEOMETRY

The new SDO_UTIL.TO_GMLGEOMETRY function, described in Chapter 19, 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.

Interior Buffers

You can create a buffer inside a polygon by specifying a negative value for the distance (`dist`) parameter with the SDO_GEOM.SDO_BUFFER function, which is described in Chapter 13.

Tablespace for Temporary Tables During Index Creation

You can specify the tablespace for temporary tables used in to create a spatial R-tree index by using the new `work_tablespace` parameter in the CREATE INDEX statement, which is described in Chapter 10.

Separate Index Table for Nonleaf Nodes

You can create a separate index table (with a name in the form MDNT_...$) for nonleaf nodes of a spatial index by specifying `'sdo_non_leaf_tbl=TRUE'` in the CREATE INDEX statement, which is described in Chapter 10. Specifying this parameter can help query performance with large data sets, and it can help overall performance for large databases where buffer pool resources are limited.

MDSYS No Longer Needed with Spatial Data Types

Public synonyms have been created for all Spatial data types; therefore, you no longer need to specify MDSYS with the data type. For example, you can declare a geometry object as type SDO_GEOMETRY instead of MDSYS.SDO_GEOMETRY.

However, you still need to specify MDSYS for the Spatial index type (`CREATE INDEX ... INDEXTYPE IS MDSYS.SPATIAL_INDEX`) and for Spatial tables (such as the MDSYS.SDO_DIST_UNITS table).

DBA_SDO_xxx Views No Longer Provided

The DBA_SDO_GEOM_METADATA, DBA_SDO_INDEX_INFO, and DBA_SDO_INDEX_METADATA views are no longer provided. You can instead use the ALL_SDO_GEOM_METADATA, ALL_SDO_INDEX_INFO, and ALL_SDO_INDEX_METADATA views, respectively. These views are described in Section 2.4 and Section 2.5.1.

SDO_MIGRATE Procedures

The following SDO_MIGRATE package procedures are no longer documented in Chapter 17:

- SDO_MIGRATE.TO_734
- SDO_MIGRATE.TO_81X
- SDO_MIGRATE.FROM_815_TO_81X

You should use the SDO_MIGRATE.TO_CURRENT procedure if you need to upgrade data to the current Spatial release.

Java Client Interface

Several Java interfaces provide access to many Spatial data types and features. Section 1.11 lists the interfaces, describes the `sdoapi` interface, and explains how to find detailed reference information in Javadoc-generated API documentation.

Transportable Tablespace Support

Before Oracle Database 10*g* Release 1 (10.1), the Oracle transportable tablespace feature could not be used with tablespaces that contained any spatial indexes. Effective with Oracle Database 10*g* Release 1 (10.1), you can transport tablespaces that contain spatial indexes. However, you must call the new SDO_UTIL.PREPARE_FOR_TTS procedure just before you perform the export operation, and you must call it for each user that has data in the specified tablespace; and you must also call the new SDO_UTIL.INITIALIZE_INDEXES_FOR_TTS procedure just after you perform the import operation. Both procedures are described in Chapter 19.

New Schema: MDDATA

The new MDDATA schema is recommended for storing data used by geocoding and routing applications. This is the default schema for Oracle software that accesses geocoding and routing data. The MDDATA schema is described in Section 1.13.

Complex Query Examples

A new appendix (Appendix C) 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.

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 methods, 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:

1 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:

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 components:

- A schema (MDSYS) that prescribes the storage, syntax, and semantics of supported geometric data types
- A spatial indexing mechanism
- A set of operators and functions for performing area-of-interest queries, spatial join queries, and other spatial analysis operations
- Administrative utilities

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 system, as explained in Section 6.5.)

1.2 Object-Relational Model

Spatial supports the **object-relational** model for representing geometries. The object-relational model uses a table with a single column of SDO_GEOMETRY and a single row per geometry instance. 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 row and 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:

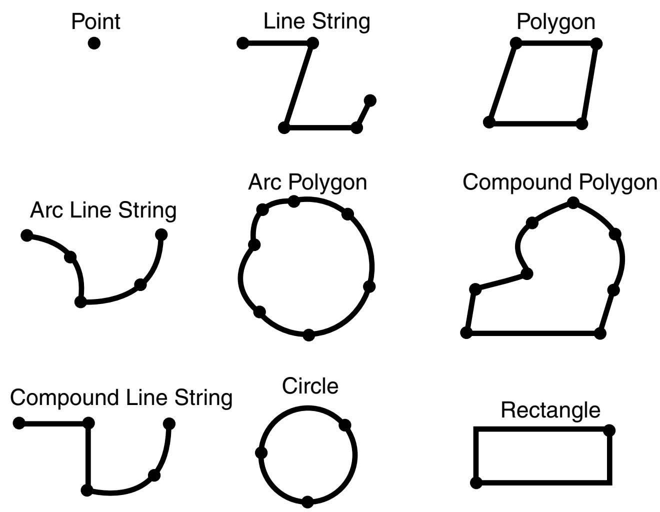
- Points and point clusters
- Line strings
- *n*-point polygons
- Arc line strings (All arcs are generated as circular arcs.)
- Arc polygons
- Compound polygons
- Compound line strings
- Circles
- Optimized rectangles

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 the interior ring of a polygon with holes are considered as two distinct elements that together make up a complex polygon.

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). Each layer's geometries and associated spatial index 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 2.6.)

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 0.001 (1 millimeter), and in most cases it should be larger. Spatial uses 0.001 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 2.4.3) 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.

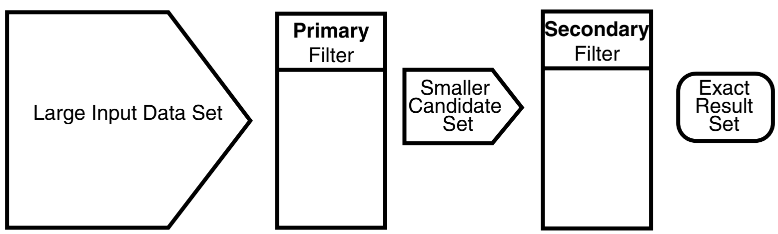
The two operations are referred to as *primary* and *secondary* filter operations.

- 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.

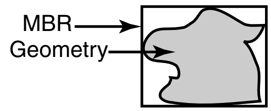
Testing of spatial indexes with many workloads and operators is ongoing, and results and recommendations will be documented as they become available.

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

1.7.1 R-Tree Indexing

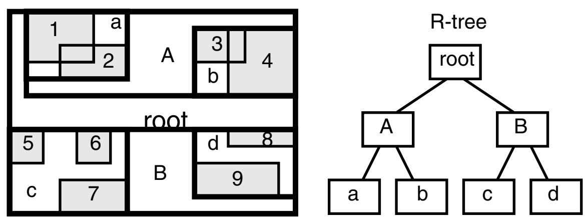
A spatial R-tree index can index spatial data of up to four dimensions. An R-tree index approximates each geometry by a single rectangle that minimally encloses the geometry (called the minimum bounding rectangle, or MBR), as shown in Figure 1–3.

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 2.5). 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 18): 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:

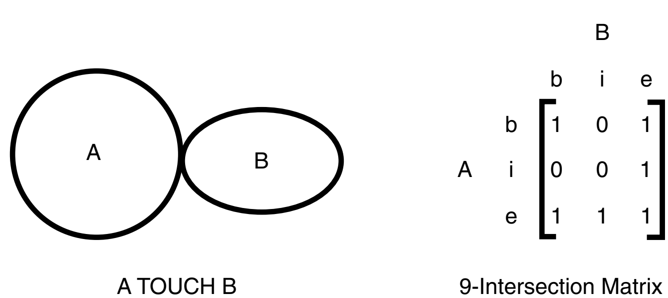
- The SDO_RELATE operator evaluates topological criteria.
- The SDO_WITHIN_DISTANCE operator determines if two spatial objects are within a specified distance of each other.
- The SDO_NN operator identifies the nearest neighbors for a spatial object.

The syntax of these operators is given in Chapter 12.

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 Ab, an interior Ai, and an exterior Ae), 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

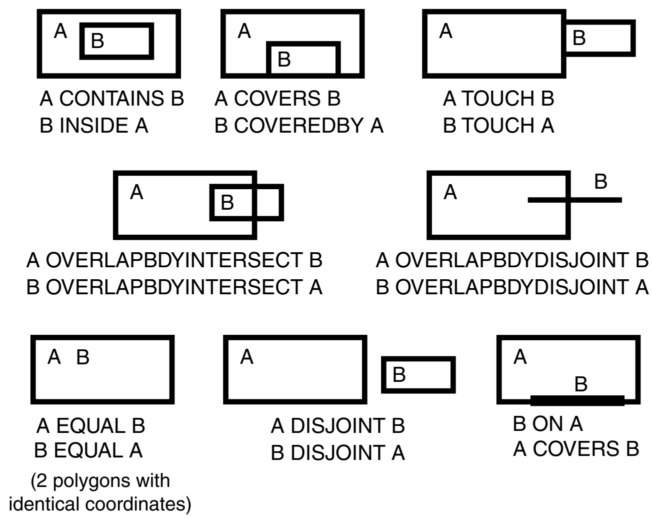


Some of the topological relationships identified in the seminal work by Professor Max Egenhofer (University of Maine, Orono) and colleagues have names associated with them. Spatial uses the following names:

- 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.

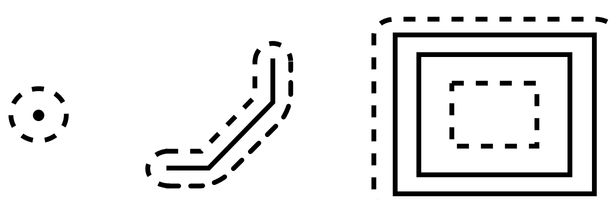
Figure 1–6 illustrates these topological relationships.

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 12.

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.

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 higher). 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 14.

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,
```

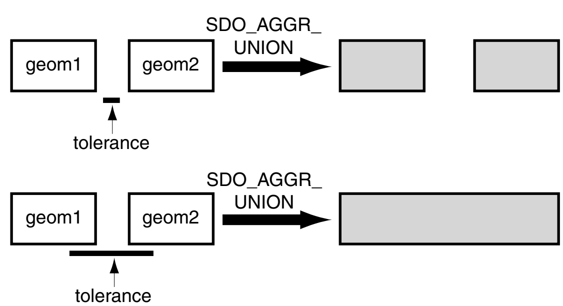
```
tolerance NUMBER);
```

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 2.4.

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 Spatial Java Interface

This section describes the `sdoapi` Java client interface for general Spatial operations. In addition to the `sdoapi` interface, Spatial provides other specialized interfaces, which are documented in other manuals:

- Topology data model client interface (`sdotopo`), described in *Oracle Spatial Topology and Network Data Models*
- Network data model client interface (`sdonm`), described in *Oracle Spatial Topology and Network Data Models*

The `sdoapi` Java client interface consists of the following classes:

- `JGeometry`: class that maps the SQL type SDO_GEOMETRY. The main methods for reading and writing database raster images are `load(STRUCT)` and `store().`
- `JGeometry.Point`: convenience class that represents a double-typed point.
- `DataException`: class for MapViewer data access exceptions.

For detailed reference information about these classes, see the Javadoc-generated API documentation: open `index.html` in a directory that includes the path `sdoapi/doc/javadoc`.

1.12 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 20.

1.13 MDDATA Schema

Effective with Oracle Database 10*g*, 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;
```

It is recommended that you 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://otn.oracle.com/products/spatial/`

Look for material relevant to Spatial performance and tuning.

1.15 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
--------------------------------------------------------------------------------
10.1.0.0.0
```

1.16 Spatial Application Hardware Requirement Considerations

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

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*.

1.17 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.

1.18 Spatial Examples

Oracle Spatial provides examples that you can use to reinforce your learning and to create models for coding certain operations. 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, as well as several functions and procedures:

- Inserting, indexing, and querying spatial data (Section 2.1)
- Coordinate systems (spatial reference systems) (Section 6.8)
- Linear referencing system (LRS) (Section 7.7)

2

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:

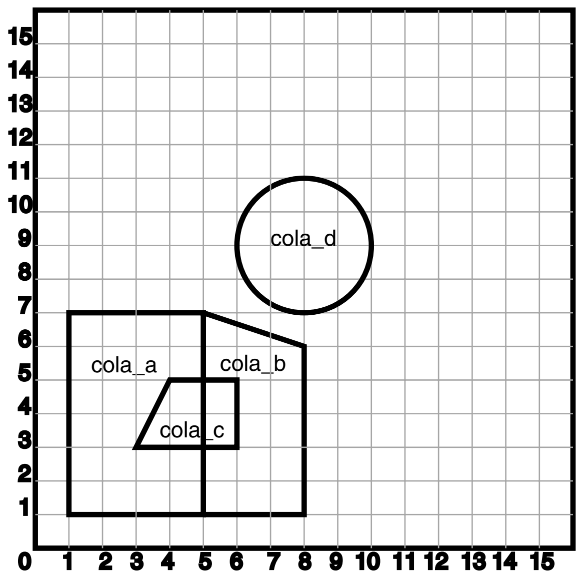
2.1 Simple Example: Inserting, Indexing, and Querying Spatial Data

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 2.4.1 and Section 2.4.2.)

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
  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;
```

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 Chapter 11.

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 and network 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
*dl*00	UNKNOWN_ GEOMETRY	Spatial ignores this geometry.
*dl*01	POINT	Geometry contains one point.
*dl*02	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.)
*dl*03	POLYGON	Geometry contains one polygon with or without holes.[1]
*dl*04	COLLECTION	Geometry is a heterogeneous collection of elements.[2] COLLECTION is a superset that includes all other types.
*dl*05	MULTIPOINT	Geometry has one or more points. (MULTIPOINT is a superset of POINT.)

Table 2–1 (Cont.) Valid SDO_GTYPE Values

Value	Geometry Type	Description
*dl*06	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.)
*dl*07	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.

[2] Polygons in the collection can be disjoint.

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 pre-release 8.1.6 format of a 1-digit SDO_GTYPE value is 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 2.4.3.

Also, if 1-digit SDO_GTYPE values are converted to 4-digit values, any SDO_ETYPE 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 Chapter 11.

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 MDSYS.CS_SRS table (described in Section 6.4.1), and this value must be inserted into the SRID column of the USER_SDO_GEOM_METADATA view (described in Section 2.4).

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 2.3.5 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. For information about LRS, see Chapter 7.

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 later in this chapter).

- 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 2.3.6.
1	1	Point type.
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 arc's start point, any point on the arc, and the arc's end point. 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.
1003 or 2003	2	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. Each circular arc is described using three coordinates: the arc's start point, any point on the arc, and the arc's end point. 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.

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

SDO_ ETYPE	SDO_ INTERPRETATION	Meaning
1003 or 2003	3	Rectangle type (sometimes called *optimized rectangle*). 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.
1003 or 2003	4	Circle type. Described by three distinct non-colinear points, all on the circumference of the circle.
4	$n > 1$	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. 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. See Section 2.3.3 and Figure 2–4 for an example of a geometry using this type.
1005 or 2005	$n > 1$	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. 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). See Section 2.3.4 and Figure 2–5 for an example of a geometry using this type.

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 2.4.

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 previously. 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 *d*001 and *d*005, any subelement not of SDO_ETYPE 1 is ignored.
- For SDO_GTYPE values *d*002 and *d*006, any subelement not of SDO_ETYPE 2 or 4 is ignored.
- For SDO_GTYPE values *d*003 and *d*007, 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.

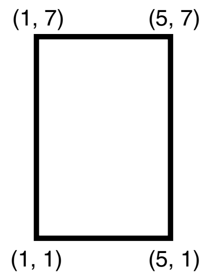
2.3 Geometry Examples

This section contains examples of several geometry types.

2.3.1 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 2–2 shows a SQL statement that inserts the geometry illustrated in Figure 2–2 into the database.

Example 2–2 SQL Statement to Insert a Rectangle

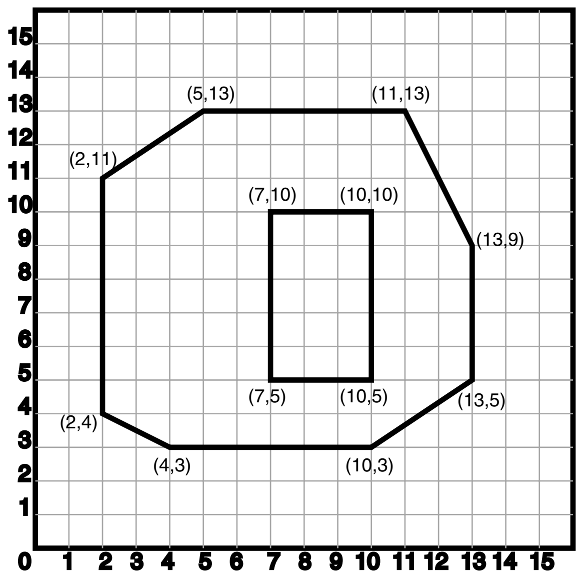
```
INSERT INTO cola_markets VALUES(
  1,
  'cola_a',
  SDO_GEOMETRY(
    2003,  -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
    SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
          -- define rectangle (lower left and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

```
);
```

2.3.2 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 2–3 shows a SQL statement that inserts the geometry illustrated in Figure 2–3 into the database.

Example 2–3 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

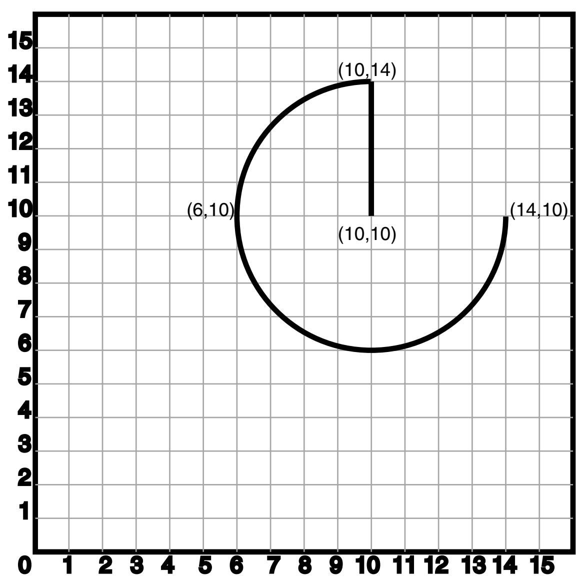
The elements in SDO_ELEM_INFO and SDO_ORDINATES must be in one of the following orders (depending on whether you specify Polygon A or Polygon B first):

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

2.3.3 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 2–4 shows a SQL statement that inserts the geometry illustrated in Figure 2–4 into the database.

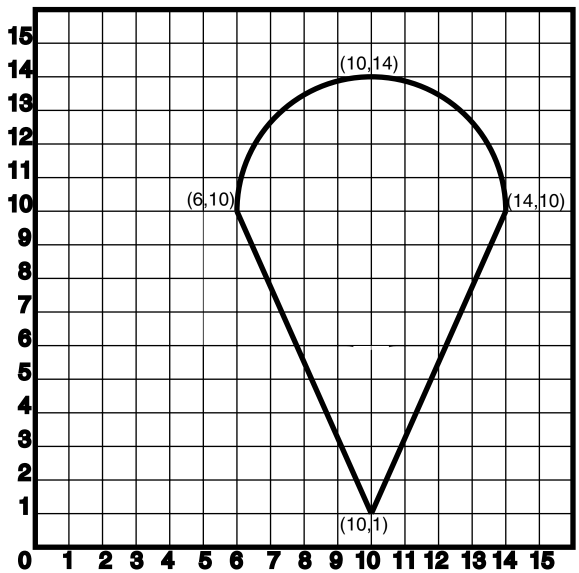
Example 2–4 SQL Statement to Insert a Compound Line String

```
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)
  )
);
```

2.3.4 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 2–5 shows a SQL statement that inserts the geometry illustrated in Figure 2–5 into the database.

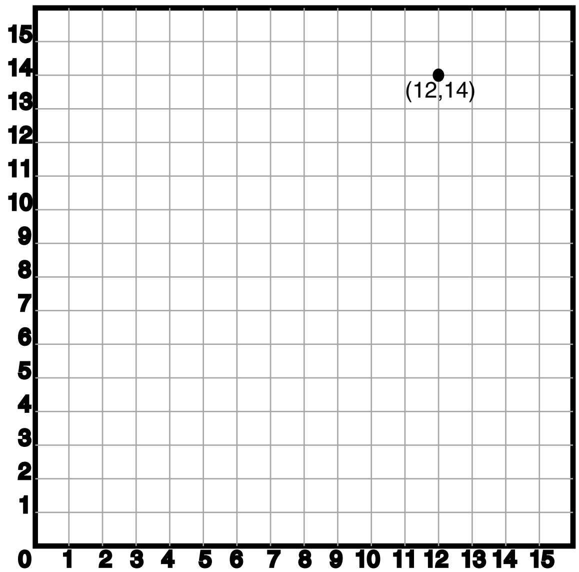
Example 2–5 SQL Statement to Insert a Compound Polygon

```
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)
  )
);
```

2.3.5 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 2–6 shows a SQL statement that inserts the geometry illustrated in Figure 2–6 into the database.

Example 2–6 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 2–7 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 2–6.

Example 2–7 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)
```

2.3.6 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)

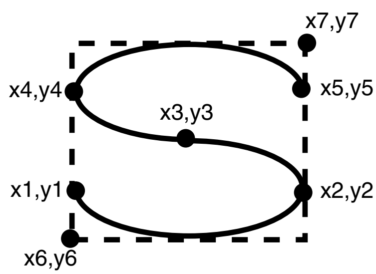
  (The SDO_INTERPRETATION value for the type 0 element can be any numeric value, and applications are responsible for determining the validity and significance of the value.)

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 2–7 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 2–7 Geometry with Type 0 (Zero) Element



In the example shown in Figure 2–7:

- 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 2–8 shows a SQL statement that inserts the geometry with a type 0 element (similar to the geometry illustrated in Figure 2–7) 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 2–8 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)
  )
);
```

2.4 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.

2.4.1 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.

2.4.2 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).

2.4.3 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, ..., X*n*, Y*n*}, 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.

2.4.4 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.)

2.5 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.

2.5.1 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.

2.5.1.1 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 2–3, 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 2–3 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 2.5.2).
SDO_INDEX_STATUS	VARCHAR2	Contains DEFERRED if the index status has been set to deferred (using the `index_status` keyword with the ALTER INDEX statement) and VALID if the index status is not deferred.

2.5.1.2 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 2–4. (The columns are listed in their order in the view definition.)

Table 2–4 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_INDEX_NAME	VARCHAR2	Name of the index.
SDO_INDEX_TABLE	VARCHAR2	Name of the spatial index table (described in Section 2.5.2).
SDO_INDEX_PRIMARY	NUMBER	Indicates if this is a primary or secondary index. 1 = primary, 2 = secondary.
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_TSNAME	VARCHAR2	Schema name of the SDO_INDEX_TABLE.
SDO_COLUMN_NAME	VARCHAR2	Name of the column on which this index is built.
SDO_INDEX_DIMS	NUMBER	Number of dimensions of the geometry objects in 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 indexed.
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_RTREE_PCTFREE	NUMBER	Minimum percentage of slots in each index tree node to be left empty when an R-tree index is created.

Table 2–4 (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_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.
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_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_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_GEODETIC	VARCHAR2	Contains TRUE if the index is geodetic (see Section 4.1.2) and FALSE if the index is not geodetic.

Table 2–4 (Cont.) Columns in the xxx_SDO_INDEX_METADATA Views

Column Name	Data Type	Purpose
SDO_INDEX_STATUS	VARCHAR2	Contains DEFERRED if the index status has been set to deferred (using the `index_status` keyword with the ALTER INDEX statement) and VALID if the index status is not deferred.

2.5.2 Spatial Index Table Definition

For an R-tree index, a spatial index table (each SDO_INDEX_TABLE entry as described in Table 2–4 in Section 2.5.1) contains the columns shown in Table 2–5.

Table 2–5 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.

2.5.3 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 2–4 in Section 2.5.1.2). 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$).

2.6 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 SDO_UNIT value from the MDSYS.SDO_DIST_UNITS or MDSYS.SDO_AREA_UNITS table. For example, 'unit=KM' in the following example (using data and definitions from Example 6–4 in Section 6.8) 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 MDSYS.SDO_DIST_UNITS and MDSYS.SDO_AREA_UNITS tables to determine which unit names are valid and what ratios to use in comparing or converting between different units.

The MDSYS.SDO_DIST_UNITS table contains the columns shown in Table 2–6.

Table 2–6 Columns in the SDO_DIST_UNITS Table

Column Name	Data Type	Purpose
SDO_UNIT	VARCHAR2	Unit string to be specified with the `unit` parameter. Examples: *M, KM, CM, MM, MILE, NAUT_MILE, FOOT, INCH*.
UNIT_NAME	VARCHAR2	Descriptive name of the unit. Examples: *Meter, Kilometer, Centimeter, Millimeter, Mile, Nautical Mile, Foot, Inch*.
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.

The MDSYS.SDO_AREA_UNITS table contains the columns shown in Table 2–7.

Table 2–7 Columns in the SDO_AREA_UNITS Table

Column Name	Data Type	Purpose
SDO_UNIT	VARCHAR2	Unit string to be specified with the `unit` parameter. Examples: *SQ_M*, *SQ_KM*, *SQ_CM*, *SQ_MM*, *SQ_MILE*, *SQ_FOOT*, *SQ_INCH*.
UNIT_NAME	VARCHAR2	Descriptive name of the unit. Examples: *Square Meter, Square Kilometer, Square Centimeter, Square Millimeter, Square Mile, Square Foot, Square Inch.*
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.

For a complete list of supported unit strings, unit names, and conversion factors, view the contents of the MDSYS.SDO_DIST_UNITS and MDSYS.SDO_AREA_UNITS tables. For example:

```
SELECT * from MDSYS.SDO_DIST_UNITS;
SELECT * from MDSYS.SDO_AREA_UNITS;
```

3

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 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 ASCII 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,
  GEOM 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 inserts into columns of type SDO_GEOMETRY. Note that 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 2–2 in Section 2.3.1
- Polygon with a hole: Example 2–3 in Section 2.3.2
- Compound polygon: Example 2–5 in Section 2.3.4

- Point: Example 2–6 and Example 2–7 in Section 2.3.5
- Type 0 (zero) element: Example 2–8 in Section 2.3.6

If a spatial index already exists on the spatial geometry table and you need to insert many rows, you can improve the performance of the insert operations by deferring spatial indexing, inserting the rows, and synchronizing the index, as explained in Section 4.1.3.

4 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 must be created on the tables for efficient access to the data. Although each spatial index can be an R-tree index or a quadtree index, you are strongly encouraged to use R-tree indexes and to avoid using quadtree indexes. 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.

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.

4.1.1 Creating R-Tree Indexes

If you create a spatial index without specifying any quadtree-specific parameters, an R-tree index is created. For example, the following statement creates a spatial R-tree index named `territory_idx` using default values for parameters that apply to R-tree indexes:

```
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.

R-tree 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 an R-tree 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 an R-tree 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 an R-tree 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.

- The R-tree index data table requires approximately 70*n bytes (where n is the number of rows in the table).

- The transient tables require up to approximately 200*n bytes (where n is the number of rows in the table); however, this space is freed up after the R-tree index is created.

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.

4.1.2 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.

4.1.3 Improving Performance with Bulk Insert Operations

If a Spatial index already exists and you need to insert many rows into the spatial geometry table, you can improve the performance of the insert operations by deferring spatial indexing, inserting the rows, and synchronizing the index. Follow these steps:

1. Modify the spatial index to set the index status to deferred. For example:

   ```
   ALTER INDEX cola_spatial_idx PARAMETERS ('index_status=deferred');
   ```

2. Insert the new rows, using the appropriate INSERT statements.
3. Synchronize the index, specifying the `sdo_batch_size` keyword with a fairly large value. For example:

   ```
   ALTER INDEX cola_spatial_idx PARAMETERS ('index_status=synchronize
      sdo_batch_size=500');
   ```

   The best value for the `sdo_batch_size` keyword is probably from 100 to 1000.

See the section about the ALTER INDEX statement in Chapter 10 for more information about the `index_status` and `sdo_batch_size` keywords.

4.1.4 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.

4.1.5 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. User B must perform the following steps:

1. Connect as user A (or have user A connect) and execute the following statement:

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

2. 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;
```

4.1.6 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 4.1.4) 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 12 for more information.

4.1.7 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 the *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 4.1.2.)
- Neither index can have a status of deferred updates. (Deferred update status is set by specifying `'index_status=deferred'` with the ALTER INDEX statement, as described in Chapter 10.)

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.

4.1.8 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, 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.)

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

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.

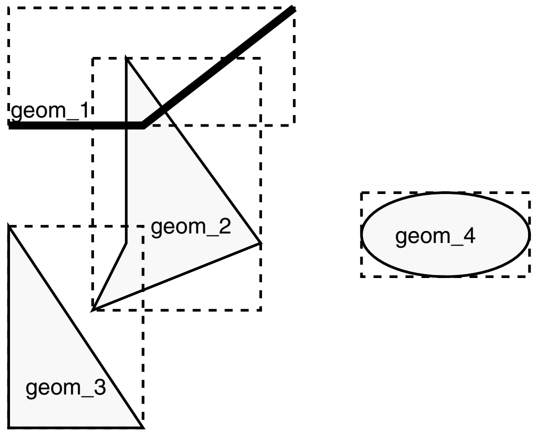
You cannot append a database link (dblink) name to the name of a spatial table in a query if a spatial index is defined on that table.

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

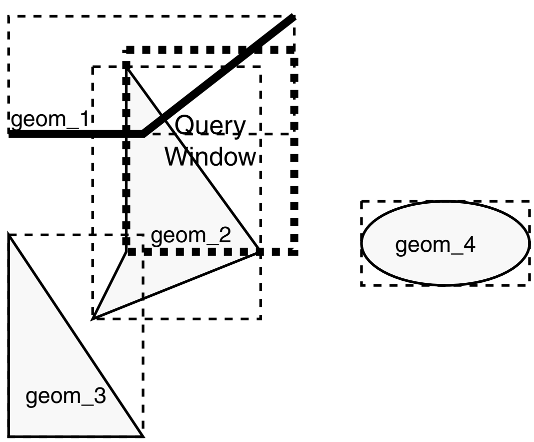
In a spatial R-tree index, each geometry is represented by its minimum bounding rectangle (MBR), as explained in Section 1.7.1. Consider the following layer containing several objects in Figure 4–1. Each object is labeled with its geometry name (geom_1 for the line string, geom_2 for the four-sided polygon, 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 query window.

4.2.1.1 Primary Filter Operator

The SDO_FILTER operator, described in Chapter 12, 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)
```

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.

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.

If the B.SHAPE column is spatially indexed with the same SDO_LEVEL value as the A.SHAPE column, the SDO_FILTER operator reuses the existing index, and performance is very good or better.

If the B.SHAPE column is spatially indexed with a different SDO_LEVEL value than the A.SHAPE column, the SDO_FILTER operator reindexes B.SHAPE in the same way as if there were no index on the column originally, and then performance is very good.

4.2.1.2 Primary and Secondary Filter Operator

The SDO_RELATE operator, described in Chapter 12, 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, as explained in the documentation for the SDO_RELATE operator in Chapter 12.

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.

If the B.SHAPE column is spatially indexed with the same SDO_LEVEL value as the A.SHAPE column, the SDO_RELATE operator reuses the existing index, and performance is very good or better.

If the B.SHAPE column is spatially indexed with a different SDO_LEVEL value than the A.SHAPE column, the SDO_RELATE operator reindexes B.SHAPE in the same way as if there were no index on the column originally, and then performance is very good.

4.2.1.3 Within-Distance Operator

The SDO_WITHIN_DISTANCE operator, described in Chapter 12, 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 12 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. The distance units are those specified in the UNIT field of the well-known text (WKT) associated with the coordinate system (in the WKTEXT column of the MDSYS.CS_SRS table, as explained in Section 6.4.1.1). 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 a.gid, b.gid FROM parks a, cosine_bufs b,
  TABLE(SDO_JOIN('PARKS', 'SHAPE', 'COSINE_BUFS', 'SHAPE',
    'mask=ANYINTERACT')) c
  WHERE c.rowid1 = a.rowid AND c.rowid2 = b.rowid;
```

4.2.1.4 Nearest Neighbor Operator

The SDO_NN operator, described in Chapter 12, 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 a keyword value pair that determines how many nearest neighbor geometries are returned by the operator. See the SDO_NN operator in Chapter 12 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 12 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 12.)

```
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 12. The following spatial join query, to list the GID column values of highways and parks where a highway interacts with a park, performs a primary filter operation only (`'mask=FILTER'`), and thus it returns only approximate results:

```
SELECT a.gid, b.gid FROM parks a, highways b,
  TABLE(SDO_JOIN('PARKS', 'SHAPE', 'HIGHWAYS', 'SHAPE',
    'mask=FILTER')) c
  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 a.gid, b.gid FROM parks a, highways b,
  TABLE(SDO_JOIN('PARKS', 'SHAPE', 'HIGHWAYS', 'SHAPE',
    'mask=ANYINTERACT')) c
  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';
```

5

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"

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).

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

Address Attribute	Description
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.
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 `match_mode` 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.
RELAX_STREET_TYPE	The street type can be different from the data used for geocoding. For example, if *Main St* is in the data used for geocoding, *Main Street* would also match that, as would *Main Blvd* if there was no *Main Blvd* and no other street type named *Main* 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 *Jones State Park* is in the data used for geocoding, *Jones State Pk* and *Jones Park* 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 *123 Main St* is in the data used for geocoding, *123 Main Lane* and *124 Main St* would also match as long as there were no ambiguities or other matches in the data.

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

Match Mode	Description
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 *Pleasant Valley* is the base name of a street in the data used for geocoding, *Pleasant Vale* 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 MATCH_CODE 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)	(Not used.)
StreetName	VARCHAR2(200)	Street name, including street type. Example: *MAIN ST*
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.

Table 5–5 (Cont.) SDO_GEO_ADDR Type Attributes

Attribute	Data Type	Description
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: *123* in *123 MAIN ST*
BaseName	VARCHAR2(200)	Base name of the street. Example: *MAIN* in *123 MAIN ST*
StreetType	VARCHAR2(20)	Type of the street. Example: ST in *123 MAIN ST*
StreetTypeBefore	VARCHAR2(1)	(Not used.)
StreetTypeAttached	VARCHAR2(1)	(Not used.)
StreetPrefix	VARCHAR2(20)	Prefix for the street. Example: *S* in *123 S MAIN ST*
StreetSuffix	VARCHAR2(20)	Suffix for the street. Example: *NE* in *123 MAIN ST NE*
Side	VARCHAR2(1)	Side of the street (`L` for left or `R` for right) that the house is on when you are traveling from lower to higher numbered addresses.
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 from lower to higher numbered addresses.
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('1CARLTONBGOODLETTPL','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('1CARLTONBGOODLETTPL','SANFRANCISCO
--------------------------------------------------------------------------------
PL

SELECT SDO_GCDR.GEOCODE('SCOTT',
  SDO_KEYWORDARRAY('1 Carlton B Goodlett Pl', 'San Francisco, CA  94102'),
  'US', 'RELAX_BASE_NAME').Side  RROM 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('1CARLTONBGOODLETTPL','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://otn.oracle.com/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 20:

- The SDO_GCDR.GEOCODE function geocodes an unformatted address to return an SDO_GEO_ADDR 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).

6 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, see Chapter 15.

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, "Coordinate Systems Data Structures"
- Section 6.5, "Creating a User-Defined Coordinate System"
- Section 6.6, "Coordinate System Transformation Functions"
- Section 6.7, "Notes and Restrictions with Coordinate Systems Support"
- Section 6.8, "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.

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 system 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 system support in Spatial.

6.1.6 Geodetic Datum

A **geodetic datum** is 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.

6.1.7 Authalic Sphere

An **authalic sphere** is a sphere that has the same surface area as a particular oblate ellipsoid of revolution representing the figure of the Earth.

6.1.8 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 Oracle9*i*, Spatial provides a rational and complete treatment of geodetic coordinates. Before Oracle9*i*, 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 arcs 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, and take longer to complete. In general, a Spatial operation using geodetic coordinates will take two to three times longer than the same operation 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 6–4 in Section 6.8):

```
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 near the North and South Poles).

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 13) 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.
- No line element can have a length longer than 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 an element representing the entire ocean surface of the Earth; however, you can create multiple elements, each representing part of the overall ocean surface.

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 higher 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 0.005 for geodetic data, this is interpreted as precise to 5 millimeters. 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 6–4 in Section 6.8) specifies these extents (with a 10-meter tolerance value in each dimension) for a geodetic data layer:

```
INSERT INTO USER_SDO_GEOM_METADATA
  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 6.7 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 MDSYS.CS_SRS table (described in Section 6.4.1). 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 can use these local coordinate systems only to convert coordinates in a local coordinate system from one unit of measurement to another (for example, inches to millimeters) by transforming a geometry or a layer of geometries.

6.4 Coordinate Systems Data Structures

The coordinate systems functions and procedures use information provided in the following tables supplied with Oracle Spatial:

- MDSYS.CS_SRS (see Section 6.4.1) defines 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 GIS Consortium (`http://www.opengis.org`).
- MDSYS.SDO_ANGLE_UNITS (see Section 6.4.2) defines the valid angle units. The angle unit is part of the well-known text description.
- MDSYS.SDO_DIST_UNITS (see Table 2–6 in Section 2.6) defines the valid distance units. The distance unit is included in the well-known text description.
- MDSYS.SDO_DATUMS (see Section 6.4.3) defines the valid datums. The datum is part of the well-known text description.
- MDSYS.SDO_ELLIPSOIDS (see Section 6.4.4) defines the valid ellipsoids. The ellipsoid (SPHEROID specification) is part of the well-known text description.
- MDSYS.SDO_PROJECTIONS (see Section 6.4.5) defines the valid map projections. The map projection is part of the well-known text description.

Note: You should not modify or delete any Oracle-supplied information in any of the tables that are used for coordinate system support.

You should not add any information to the MDSYS.CS_SRS table unless you are creating a user-defined coordinate system. (Do not add information to the MDSYS.SDO_DATUMS, MDSYS.SDO_ELLIPSOIDS, or MDSYS.PROJECTIONS tables.) Section 6.5 describes how to create a user-defined coordinate system.

6.4.1 MDSYS.CS_SRS Table

The MDSYS.CS_SRS reference table contains over 900 rows, one for each valid coordinate system.

Note: You should probably not modify, delete, or add any information in the MDSYS.CS_SRS table. If you do plan to modify this table, you should connect to the database as the MDSYS user.

If you plan to add any user-defined coordinate systems, be sure to use SRID values of 1000000 (1 million) or higher, and follow the guidelines in Section 6.5.

The MDSYS.CS_SRS table contains the columns shown in Table 6–1.

Table 6–1 MDSYS.CS_SRS Table

Column Name	Data Type	Description
CS_NAME	VARCHAR2(68)	A well-known name, often mnemonic, by which a user can refer to the coordinate system.
SRID	NUMBER(38)	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.
AUTH_SRID	NUMBER(38)	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.
AUTH_NAME	VARCHAR2(256)	An authority name for the coordinate system. Contains 'Oracle' in the supplied table. Users can specify any value in any rows that they add.
WKTEXT	VARCHAR2(2046)	The well-known text (WKT) description of the SRS, as defined by the Open GIS Consortium. For more information, see Section 6.4.1.1.
CS_BOUNDS	SDO_GEOMETRY	An optional SDO_GEOMETRY 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.

6.4.1.1 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 GIS Consortium.

The following is the WKT EBNF syntax. All user-defined coordinate systems must strictly comply with this 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 ]
```

The prime meridian (PRIMEM) must be 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 15, to validate the WKT of any coordinate system defined in the MDSYS.CS_SRS table.

6.4.2 MDSYS.SDO_ANGLE_UNITS Table

The MDSYS.SDO_ANGLE_UNITS reference table contains one row for each valid UNIT specification in the well-known text (WKT) description in the coordinate system definition. The WKT is described in Section 6.4.1.1.

The MDSYS.SDO_ANGLE_UNITS table contains the columns shown in Table 6–2.

Table 6–2 MDSYS.SDO_ANGLE_UNITS Table

Column Name	Data Type	Description
SDO_UNIT	VARCHAR2(32)	(Reserved for future use by Oracle Spatial.)
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.

6.4.3 MDSYS.SDO_DATUMS Table

The MDSYS.SDO_DATUMS reference table contains 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 6.4.1.1.

The MDSYS.SDO_DATUMS table contains the columns shown in Table 6–3.

Table 6–3 MDSYS.SDO_DATUMS Table

Column Name	Data Type	Description
NAME	VARCHAR2(64)	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 + (SCALE_ADJUST * 10^{-6})

The following are the names (in tabular format) of the supported datums:

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		

6.4.4 MDSYS.SDO_ELLIPSOIDS Table

The MDSYS.SDO_ELLIPSOIDS reference table contains 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 6.4.1.1.

The MDSYS.SDO_ELLIPSOIDS table contains the columns shown in Table 6–4.

Table 6–4 MDSYS.SDO_ELLIPSOIDS Table

Column Name	Data Type	Description
NAME	VARCHAR2(64)	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 supported ellipsoids:

Airy 1930	Airy 1930 (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	

6.4.5 MDSYS.SDO_PROJECTIONS Table

The MDSYS.SDO_PROJECTIONS reference table contains 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 6.4.1.1.

The MDSYS.SDO_PROJECTIONS table contains the column shown in Table 6–5.

Table 6–5 MDSYS.SDO_PROJECTIONS Table

Column Name	Data Type	Description
NAME	VARCHAR2(64)	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 supported projections:

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

6.5 Creating a User-Defined Coordinate System

To create a user-defined coordinate system, add a row to the MDSYS.CS_SRS table. See Section 6.4.1 for information about this table, including the requirements for values in each column.

To specify the WKTEXT column in the MDSYS.CS_SRS table, follow the syntax specified in Section 6.4.1.1. See also the examples in that section.

When you specify the WKTEXT column entry, use valid values from several Spatial reference tables:

- MDSYS.SDO_ANGLE_UNITS (see Section 6.4.2) in a UNIT specification for angle units
- MDSYS.SDO_DIST_UNITS (see Table 2–6 in Section 2.6) in a UNIT specification for distance units
- MDSYS.SDO_DATUMS (see Section 6.4.3) in the DATUM specification, or a user-defined datum not in MDSYS.SDO_DATUMS

  If you supply a user-defined datum, the datum name must be different from any datum name in the MDSYS.SDO_DATUMS table, and the WKT must specify at least the datum name and the spheroid (or ellipsoid) information listed in Section 6.4.1.1. If the shift, rotation, and scale parameters are all zero, you can omit them; however, if any of these parameter values are nonzero, you must specify them all.
- MDSYS.SDO_ELLIPSOIDS (see Section 6.4.4) in the SPHEROID specification

  If you supply a user-defined ellipsoid, the ellipsoid name must be different from any ellipsoid name in the MDSYS.SDO_ELLIPSOIDS table. You must also specify the semi-major axis and inverse flattening for a user-defined ellipsoid.
- MDSYS.SDO_PROJECTIONS (see Section 6.4.5) in the PROJECTION specification

The name in each PARAMETER specification must be one of the following, depending on the projection that you use:

- `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 meters)
- `False_Northing` (in meters)
- `Perspective_Point_Height` (in meters)
- `Landsat_Number` (must be 1, 2, 3, 4, or 5)
- `Path_Number`
- `Scale_Factor`

Some of these parameters are appropriate for several projections. They are not all appropriate for every projection.

Example 6–2 creates a user-defined projected coordinate system. The first four columns are not the WKT information, but specify other fields in the MSDYD.CS_SRS table. The WKT information starts with PROJCS. This example is similar to an existing coordinate system, but has a different name, SRID, and central meridian.

Example 6–2 Creating a User-Defined Projected Coordinate System

```
INSERT INTO mdsys.cs_srs VALUES ('UTM Zone 44.5, Northern Hemisphere (WGS 84)',
1082378, 1082378, 'Oracle',
'PROJCS["UTM Zone 44.5, Northern Hemisphere (WGS 84)",
GEOGCS [ "WGS 84",
DATUM ["WGS 84 ",
SPHEROID ["WGS 84", 6378137.000000, 298.257224]],
PRIMEM [ "Greenwich", 0.000000 ],
UNIT ["Decimal Degree", 0.01745329251994330]],
PROJECTION ["Transverse Mercator"],
PARAMETER ["Scale_Factor", 0.999600],
PARAMETER ["Central_Meridian", 84.000000],
PARAMETER ["False_Easting", 500000.000000],
UNIT ["Meter", 1.000000000000]]',NULL);
```

Example 6–3 creates a user-defined geodetic coordinate system. The first four columns are not the WKT information, but specify other fields in the MSDYD.CS_SRS table. The WKT information starts with GEOGCS. This example includes an ellipsoid (SPHEROID) definition in which the semi-major axis and inverse flattening parameters are slightly changed from the WGS 84 coordinate system, as well as a different datum definition. Because the `shift_x` and `shift_y` parameter values are specified, all the shift, rotation, and scaling values must be specified. There is no projection information included for a geodetic coordinate system.

Example 6–3 Creating a User-Defined Geodetic Coordinate System

```
INSERT INTO mdsys.cs_srs  VALUES
( 'Longitude / Latitude (WGS 90)', 1008307, 1008307, 'Oracle',
'GEOGCS [ "Longitude / Latitude (WGS 90)",
DATUM ["WGS 90",
SPHEROID ["WGS 90", 6378137.032499, 298.257236], 100, 100, 0, 0, 0, 0, 0],
PRIMEM [ "Greenwich", 0.000000 ],
UNIT ["Decimal Degree", 0.01745329251994330]]',NULL);
```

6.6 Coordinate System Transformation Functions

The current release of Oracle Spatial includes the following functions and procedures for data transformation using coordinate systems:

- SDO_CS.TRANSFORM function: Transforms a geometry representation using a coordinate system (specified by SRID or name).
- SDO_CS.TRANSFORM_LAYER procedure: Transforms an entire layer of geometries (that is, all geometries in a specified column in a table).
- SDO_CS.VALIDATE_WKT function: Validates the well-known text (WKT) description associated with a specified SRID.
- SDO_CS.VIEWPORT_TRANSFORM function: Transforms an optimized rectangle into a valid polygon for use with Spatial operators and functions.

Reference information about these functions and procedures is in Chapter 15.

Support for additional functions and procedures is planned for future releases of Oracle Spatial.

6.7 Notes and Restrictions with Coordinate Systems Support

The following notes and restrictions apply to coordinate systems support in the current release of Spatial.

If you have geodetic data, see also Section 6.2 for considerations, guidelines, and additional restrictions.

6.7.1 Different Coordinate Systems for Geometries with Operators and Functions

For Spatial operators (described in Chapter 12) 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 13) that take two geometries as input parameters, both geometries must be based on the same coordinate system.

6.7.2 Functions Not Supported with Geodetic Data

In the current release, the following functions are not supported with geodetic data:

- SDO_GEOM.SDO_MAX_MBR_ORDINATE
- SDO_GEOM.SDO_MIN_MBR_ORDINATE
- All *3D* formats of LRS functions (explained in Section 7.4)

6.7.3 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.

6.8 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 15.

Example 6–4 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 6–4 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 6–4 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 6–5 includes the output of the SELECT statements in Example 6–4.

Example 6–4 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);

-- Note about areas of interest: cola_a (rectangle) and
-- cola_b (four-sided polygon) are side by side (share one border).
-- cola_c is a small four-sided polygon that overlaps parts of
-- cola_a and cola_b. A rough sketch:
--      ---------+
--      |    a   |  b   \
--      |     +------+     |
--      |   /___c____|     |
--      |        |         |
--      ---------+---------|

-- 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
  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_srname 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 6–5 shows the output of the SELECT statements in Example 6–4. 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 6–5 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_srname 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_ARR
AY(5.00069354, 1.00274488, 8.00062191, 1.00274427, 8.00062522, 6.00315345, 5.000
6986, 7.00323528, 5.00069354, 1.00274488))

SDO_ROWID
------------------
GEOMETRY(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
```

```
AAABZzAABAAAOa6AAC
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))

AAABZzAABAAAOa6AAD
SDO_GEOMETRY(2003, 8199, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1), SDO_ORDINATE_ARR
AY(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
```

7 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 systems.

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"

7.1 Terms and Concepts

This section explains important terms and concepts related to linear referencing support in Oracle Spatial.

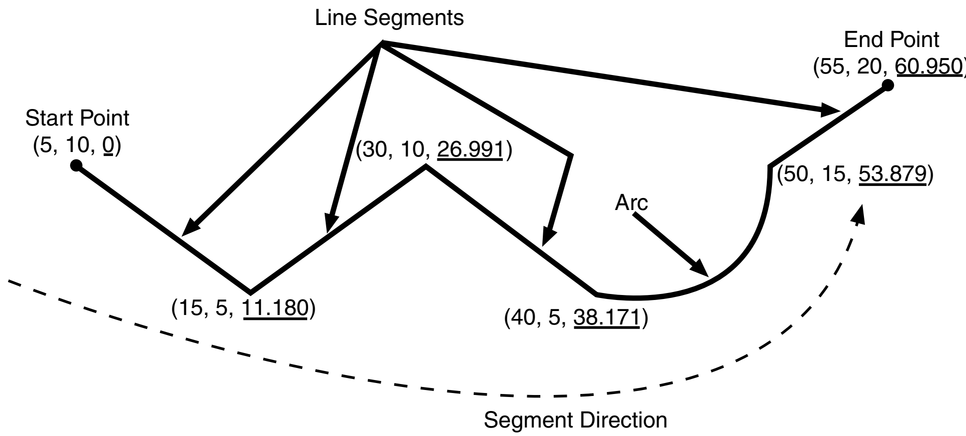
7.1.1 Geometric Segments (LRS Segments)

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.

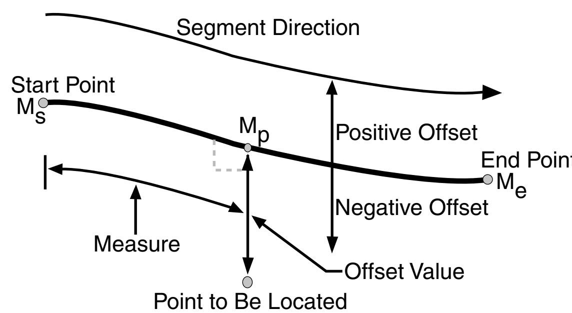
7.1.5 Offset

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

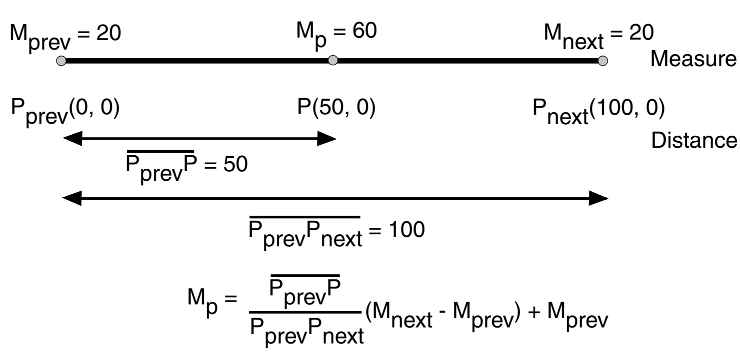
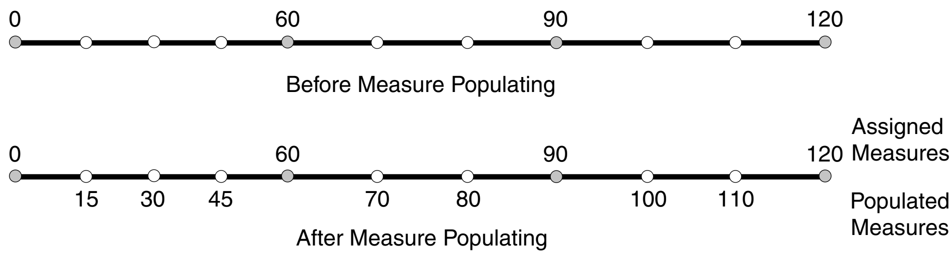


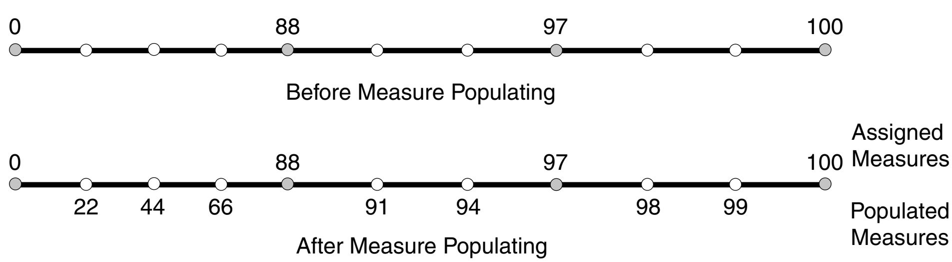
Figure 7–4 Measure Populating of a Geometric Segment



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

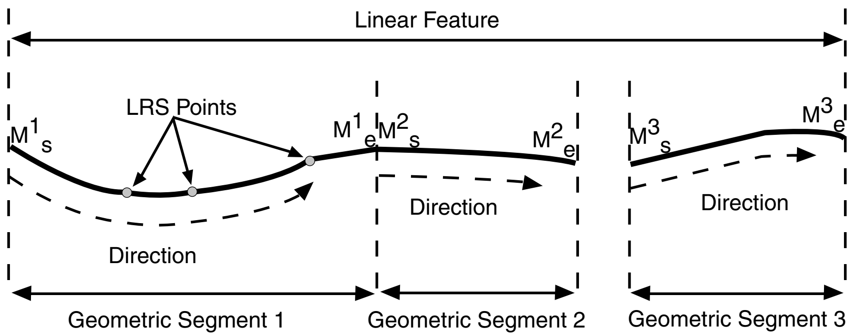
LRS points are points with linear measure information along a geometric segment. A valid LRS point is a point geometry with measure information.

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



7.2 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.

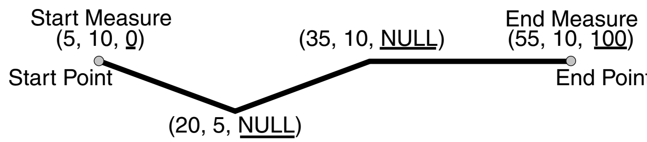
Oracle Spatial LRS support affects the Spatial metadata and data (the geometries). Example 7–1 shows how a measure dimension can be added to two-dimensional geometries in the Spatial metadata. The measure dimension must be the last element of the SDO_DIM_ARRAY in a spatial object definition (shown in bold in Example 7–1).

Example 7–1 Including LRS Measure Dimension in Spatial Metadata

```
INSERT INTO user_sdo_geom_metadata VALUES(
  'LRS_ROUTES',
  'GEOMETRY',
  SDO_DIM_ARRAY (
    SDO_DIM_ELEMENT('X', 0, 100, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 100, 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 higher format, specifically with 4-digit SDO_GTYPE and SDO_ETYPE values. For example, in Oracle Spatial release 8.1.6 and higher, 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 pre-release 8.1.6 format 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 12 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.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;
```

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.

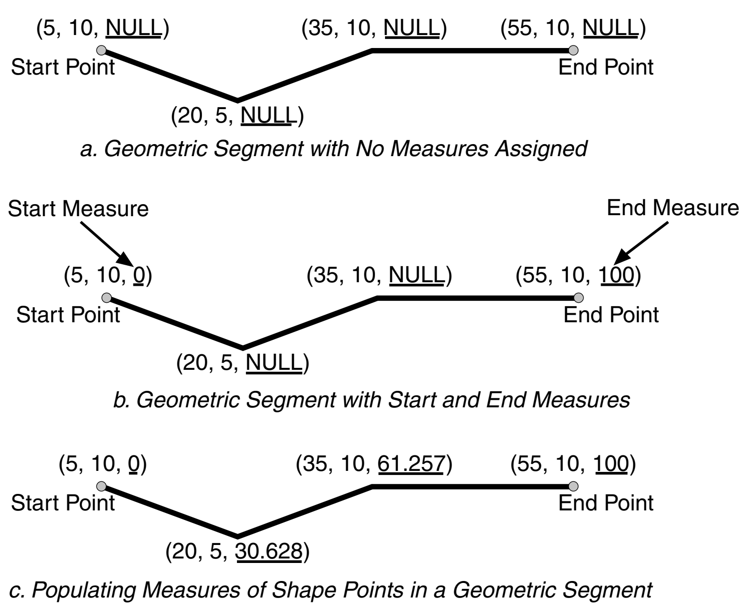
7.5.1 Defining a Geometric Segment

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/or 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.

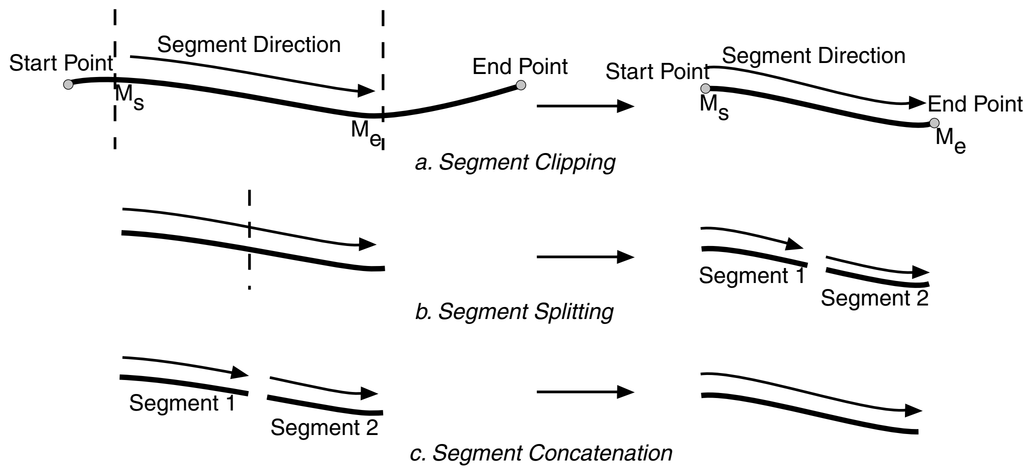
Figure 7–9 Redefining a Geometric Segment

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.

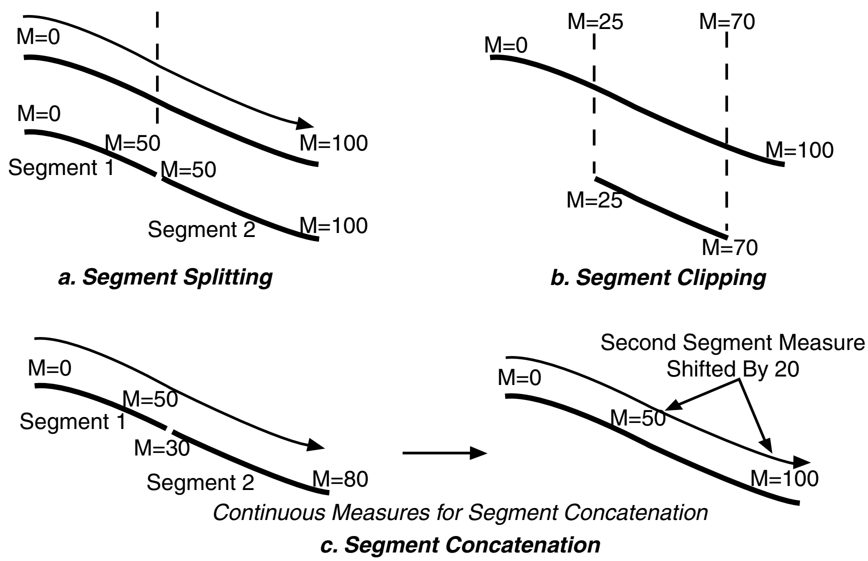
Note: In Figure 7–10 and several figures that follow, small gaps between segments are used in illustrations of segment splitting and concatenation. Each gap simply reinforces the fact that two different segments are involved. However, the two segments (such as segment 1 and segment 2 in Figure 7–10, 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. 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.

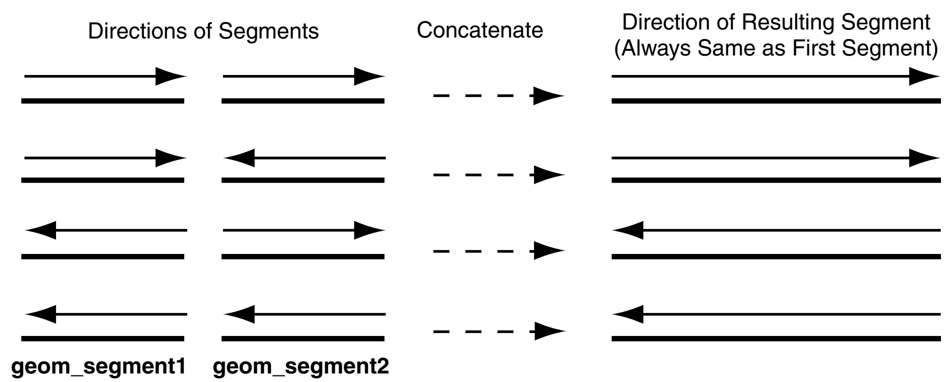
Measure assignments for the clipping, splitting, and concatenating operations in Figure 7–10 are shown in Figure 7–11. Measure information and segment direction are preserved in a consistent manner. The assignment is done automatically when the operations have completed.

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

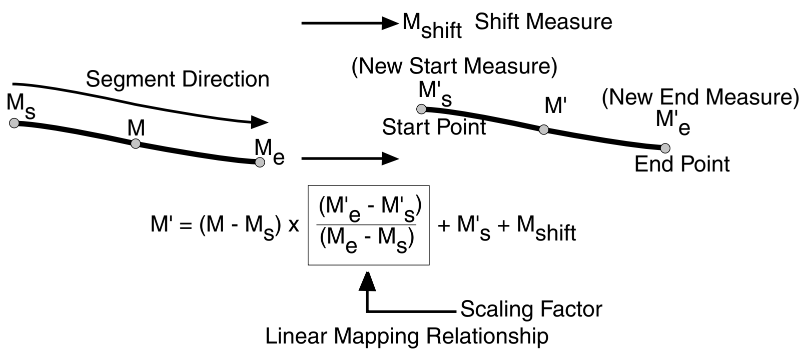


In addition to explicitly concatenating two connected segments using the SDO_LRS.CONCATENATE_GEOM_SEGMENTS function, you can perform aggregate concatenation: that is, you can concatenate all connected geometric segments in a column (layer) using the SDO_AGGR_LRS_CONCAT spatial aggregate function. (See the description and example of the SDO_AGGR_LRS_CONCAT spatial aggregate function in Chapter 14.)

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:

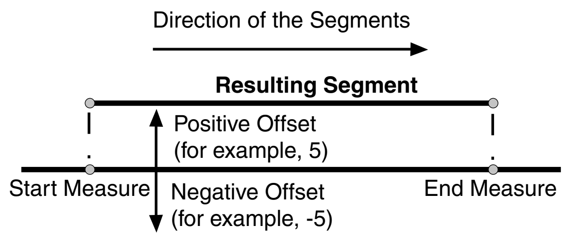
- Translating (shifting) measure information. (For example, add the same value to Ms and Me to get M's and M'e.)
- Reversing measure information. (Let M's = Me, M'e = Ms, and Mshift = 0.)
- Performing simple scaling of measure information. (Let Mshift = 0.)

For examples of these operations, see the Usage Notes and Examples for the SDO_LRS.SCALE_GEOM_SEGMENT function 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



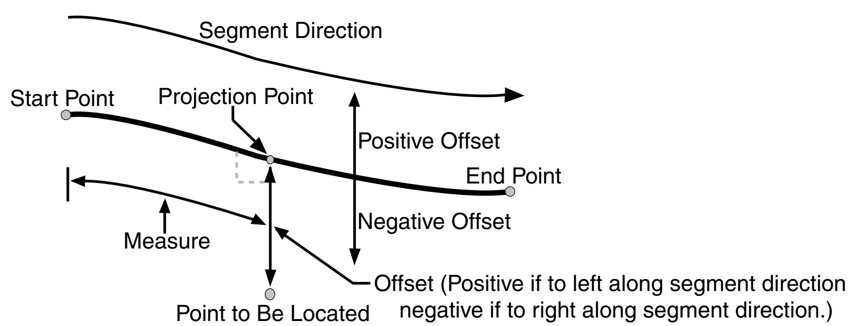
In the offsetting operation shown in Figure 7–14, the resulting geometric segment is offset by 5 units from the specified start and end measures of the original segment.

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

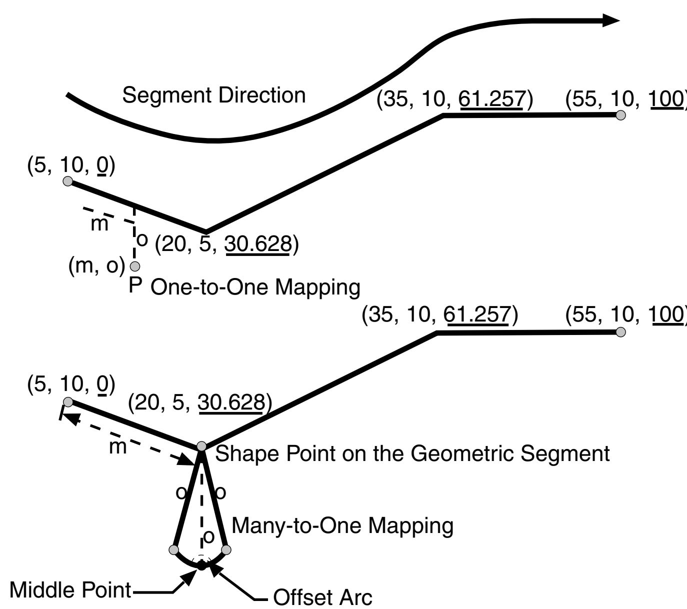
You can find the position of a point described by a measure and an offset on a geometric segment (see Figure 7–15).

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



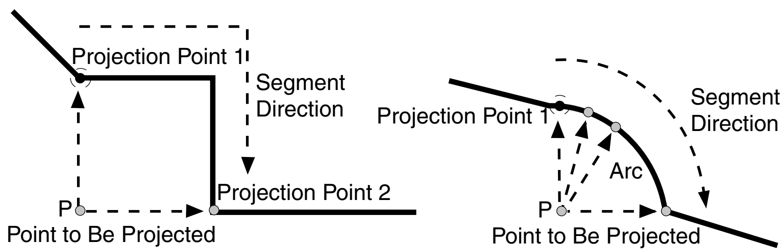
As shown in Figure 7–16, an offset arc of a shape point on a geometric segment is an arc on which all points have the same minimum distance to the shape point. As a result, all points on the offset arc are represented by the same (measure, offset) pair. To resolve this one-to-many mapping problem, the 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)

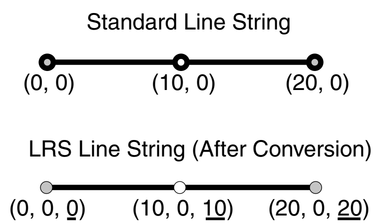
  For conversion from standard format to LRS format, a measure dimension (named *M* by default) is added, but no measure information is provided for each point. For conversion from LRS format to standard format, the measure dimension and information are removed. In both cases, the dimensional information (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.

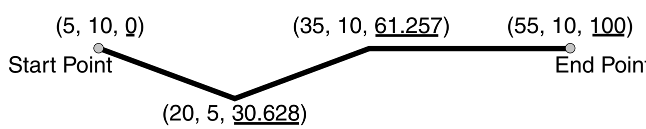
7.6 Tolerance Values with LRS Functions

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 and/or end point of the returned segment, the shape point is used as the start point and/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



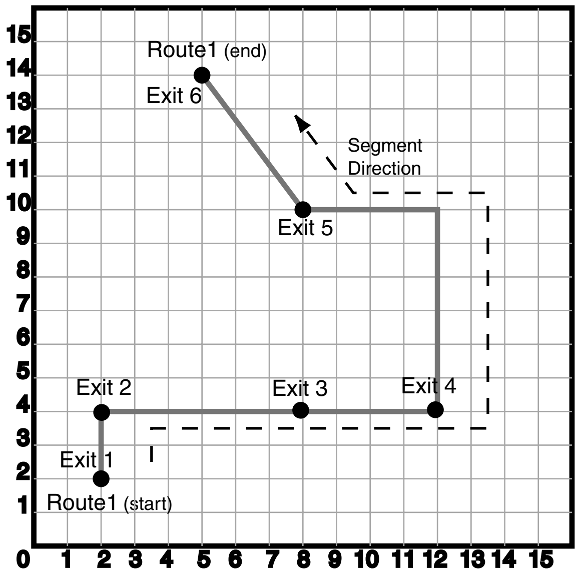
If you request a clip operation to return the segment between measure values 0 (the start point) and 61.5 in Figure 7–19, and if the distance between the points associated with measure values 61.5 and 61.257 is less than the 0.5 tolerance value, the end point of the returned segment is (35, 10, 61.257).

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

Table 7–1 (Cont.) Highway Features and LRS Counterparts

Highway Feature	LRS Feature
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
- Inserts the definition of the highway 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
  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 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 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))
```

8 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 21.

This chapter contains the following major sections:

8.1 Spatial Information and Data Mining Applications

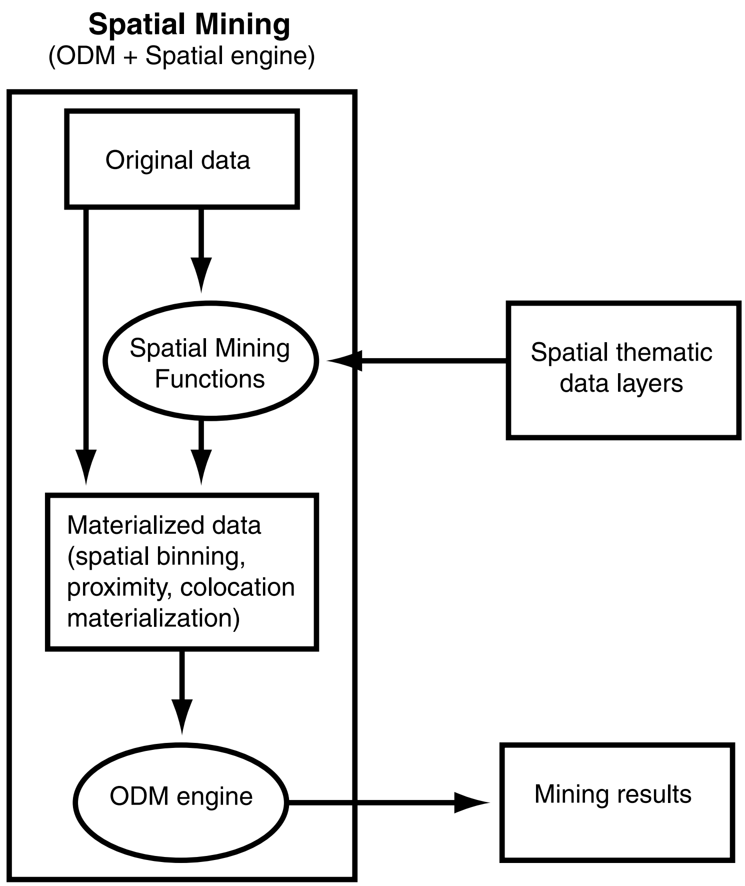
ODM allows automatic discovery of knowledge from a database. Its techniques include discovering hidden associations between different data attributes, classification of data based on some samples, and clustering to identify intrinsic patterns. For example, ODM might enable you to discover that sales prospects with high incomes are more likely to watch a particular television program or to respond favorably to a particular advertising solicitation.

Effective with Oracle Database 10*g*, spatial data can be materialized for inclusion in data mining applications. For example, 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 mining 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 21, 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 21, perform operations related to materializing spatial correlation:

- SDO_SAM.SIMPLIFY_GEOMETRY
- 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 colocated with video stores than at restaurants not colocated 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 colocated in animal habitats, and high-sales pizza restaurants could be colocated 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 colocated 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 21, 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 21, 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.

9 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 2.4.2) in the metadata view.

```
INSERT INTO USER_SDO_GEOM_METADATA
  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*

See especially the information in those documents about requirements and restrictions related to function-based indexes. For example, you must grant Spatial application users the QUERY REWRITE privilege, and you must have the initialization parameters QUERY_REWRITE_ENABLED=TRUE and QUERY_REWRITE_INTEGRITY=TRUSTED.

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 2.4.2) 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 2.4.2) 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 methods, 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, "SDO_GEOMETRY Object Type Methods"
- Chapter 12, "Spatial Operators"
- Chapter 13, "Geometry Subprograms"
- Chapter 14, "Spatial Aggregate Functions"
- Chapter 15, "Coordinate System Transformation Subprograms"
- Chapter 16, "Linear Referencing Subprograms"
- Chapter 17, "SDO_MIGRATE Procedure"
- Chapter 18, "Spatial Tuning Subprograms"
- Chapter 19, "Spatial Utility Subprograms"
- Chapter 20, "Geocoding Subprograms"
- Chapter 21, "Spatial Analysis and Mining Subprograms"

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".

10

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
INDEX_PARAMS	Allows you to change the characteristics of the spatial index.
index_status	Specifies that index modifications are to be deferred (`'index_status=deferred'`) or that deferred index modifications are to be synchronized with the data in the spatial table (`'index_status=synchronize'`). See the Usage Notes for further details. Data type is VARCHAR2.
sdo_batch_size	Specifies the number of rows to be processed at a time when the index is synchronized (`'index_status=synchronize'`). See Section 4.1.3 for more information about using this keyword to improve performance when many rows need to be inserted. Data type is NUMBER. For example: 'sdo_batch_size=500'
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 12 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. 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.
PHYSICAL_STORAGE_ PARAMS	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.
{ NOPARALLEL | PARALLEL [integer] }	Controls whether serial execution (NOPARALLEL) or parallel (PARALLEL) 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 PARALLEL keyword. Default = NOPARALLEL. (If PARALLEL 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

This statement is used to change the parameters of an existing index. This is the only way you can add or build multiple indexes on the same column.

The `index_status` keyword lets you defer modifications to the spatial index when geometries are inserted, updated, or deleted in a spatial table. Deferring the index modifications allows the geometry insert, update, and delete operations to be completed sooner, and it can reduce concurrency issues with R-tree indexes if multiple sessions are inserting rows into the spatial table. While index modifications are being deferred, spatial functions and procedures will work correctly with the current table data; however, spatial operator-based queries might perform more slowly, will not include the results of new insert operations, and might not include the results of new update operations. Therefore, you are advised not to use spatial operators while index modifications are being deferred.

For partitioned indexes, the index status can only be changed for a single partition at a time. That is, you cannot set all index partitions to deferred status with a single ALTER INDEX statement.

If you set the index status to `deferred`, you must later specify `index_status=synchronize` to make the index reflect the data in the table and to set the index to a valid state. Another use of `index_status=synchronize` is to return the index to a consistent state if an attempt to commit or roll back a transaction failed due to insufficient resources.

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 and the SDO_LEVEL value for partition IP2 of the spatial index named BGI.

```
ALTER INDEX bgi MODIFY PARTITION ip2
   PARAMETERS ('tablespace=TBS_3 sdo_level=4');
```

The following example defers index modifications and later (after the updates to the spatial table) synchronizes the index to reflect the table.

```
ALTER INDEX xyz_idx PARAMETERS ('index_status=deferred');
   .
   . <Insert rows in spatial table.>
   .
```

```
ALTER INDEX xyz_idx PARAMETERS ('index_status=synchronize');
```

The following example defers index modifications for an index partition and later (after the updates to the spatial table) synchronizes the index partition to reflect the table.

```
ALTER INDEX part_sidx MODIFY PARTITION p3
   PARAMETERS ('index_status=deferred');
   .
   . <Insert rows in spatial table.>
   .
ALTER INDEX part_sidx MODIFY PARTITION p3
   PARAMETERS ('index_status=synchronize');
```

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 ] }] ;

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
REBUILD_PARAMS	Specifies in a command string the index parameters to use in rebuilding the spatial 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.
rebuild_index	Specifies the name of the spatial index table to be rebuilt. Data type is VARCHAR2.
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 12 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.
{ NOPARALLEL | PARALLEL [integer] }	Controls whether serial execution (NOPARALLEL) or parallel (PARALLEL) 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 PARALLEL keyword. Default = NOPARALLEL. (If PARALLEL 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.

This 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 4.1.4.

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 with an SDO_LEVEL value of 12.

```
ALTER INDEX oldindex REBUILD PARAMETERS('sdo_level=12');
```

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_name> ON [schema.]<tableName> (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
INDEX_PARAMS	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_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 12 cannot be used. Data type is NUMBER. Default = 2.

Value	Description
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
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 creating the spatial index data table. A spatial index data table is a regular Oracle table with a prescribed format. Not all physical_storage_params 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.
work_tablespace	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.)

Value	Description
{ NOPARALLEL | PARALLEL [integer] }	Controls whether serial execution (NOPARALLEL) 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 4.1.1.

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 4.1.6 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 PARALLEL clause is not supported for adding an index table with the ALTER INDEX statement; however, it is supported for rebuilding such an index table with the ALTER INDEX REBUILD statement. One useful scenario is to add a small second index table, and later rebuild the index table specifying the desired parameters and using parallel execution. See the parallel execution example for the ALTER INDEX REBUILD statement.
- The performance cost and benefits from parallel execution for creating or rebuilding an index depend on a system's resources and load. If the system's 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 4.1.4.

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 4.1.2.)

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 allows you to specify a tolerance value, and specify the tolerance value in meters.

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

11

SDO_GEOMETRY Object Type Methods

This chapter contains reference and usage information for the SDO_GEOMETRY object type methods.

The SDO_GEOMETRY object type is described in Section 2.2. The type methods are listed in Table 11–1.

Table 11–1 SDO_GEOMETRY Type Methods

Method	Description
GET_DIMS	Returns the number of dimensions of a geometry object.
GET_GTYPE	Returns the geometry type of a geometry object.
GET_LRS_DIM	Returns the measure dimension of an LRS geometry object.

GET_DIMS

Format

GET_DIMS( ) RETURN NUMBER;

Description

Returns the number of dimensions of a geometry object, as specified in its SDO_GTYPE value.

Parameters

None.

Usage Notes

The SDO_TYPE value is 4 digits in the format *dltt*, as described in Section 2.2.1. This method returns the *d* (dimensionality) value, that is, the number of dimensions.

Examples

The following example returns the number of dimensions of the `cola_d` geometry object. (The example uses the definitions and data from Section 2.1.)

```
SELECT c.mkt_id, c.shape.GET_DIMS()
  FROM cola_markets c WHERE c.name = 'cola_d';

    MKT_ID C.SHAPE.GET_DIMS()
---------- ------------------
         4                  2
```

GET_GTYPE

Format

GET_GTYPE( ) RETURN NUMBER;

Description

Returns the geometry type of a geometry object, as specified in its SDO_GTYPE value.

Parameters

None.

Usage Notes

The SDO_TYPE value is 4 digits in the format *dltt*, as described in Section 2.2.1. This method returns the *tt* value, that is, the geometry type.

Examples

The following example returns the geometry type of each geometry object in the COLA_MARKETS table. (The example uses the definitions and data from Section 2.1.)

```
SELECT c.mkt_id, c.shape.GET_GTYPE() FROM cola_markets c;

    MKT_ID C.SHAPE.GET_GTYPE()
---------- -------------------
         1                   3
         2                   3
         3                   3
         4                   3
```

GET_LRS_DIM

Format

GET_LRS_DIM( ) RETURN NUMBER;

Description

Returns the measure dimension of an LRS geometry object, as specified in its SDO_GTYPE value.

Parameters

None.

Usage Notes

The SDO_TYPE value is 4 digits in the format *dltt*, as described in Section 2.2.1. This method returns the *l* value.

The *l* value is meaningful only for LRS geometry objects, and must be 0, 3, or 4:

- 0 indicates that the geometry is a pre-release 9.0.1 LRS geometry with measure as the default (last) dimension, or that the geometry is a release 9.0.1 standard geometry.
- 3 indicates that the third dimension contains the measure information.
- 4 indicates that the fourth dimension contains the measure information.

Examples

The following example returns the measure dimension of the Route 1 geometry object. (This example uses the definitions from the example in Section 7.7.)

```
SELECT a.route_id, a.route_geometry.GET_LRS_DIM()
    FROM lrs_routes a WHERE  a.route_id = 1;

  ROUTE_ID A.ROUTE_GEOMETRY.GET_LRS_DIM()
---------- ------------------------------
         1                              3
```

12

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 12–1 lists the main operators.

Table 12–1 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 12–2 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 12–2 lists operators, provided for convenience, that perform an SDO_RELATE operation of a specific mask type.

Table 12–2 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.
SDO_ON	Checks if any geometries in a table have the ON topological relationship with a specified geometry.
SDO_OVERLAPBDYDISJOINT	Checks if any geometries in a table have the OVERLAPBDYDISJOINT topological relationship with a specified geometry.
SDO_OVERLAPBDYINTERSECT	Checks if any geometries in a table have the OVERLAPBDYINTERSECT topological relationship with a specified geometry.
SDO_OVERLAPS	Checks if any geometries in a table overlap (that is, have the OVERLAPBDYDISJOINT or OVERLAPBDYINTERSECT topological relationship with) a specified geometry.
SDO_TOUCH	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.

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);

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.

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 the geometry2 column is spatially indexed with the same SDO_LEVEL value as the geometry1 column, the operator reuses the existing index, and performance is very good or better.
- If the geometry2 column is spatially indexed with a different SDO_LEVEL value than the geometry1 column, the operator reindexes geometry2 in the same way as if there were no index on the column originally, and then 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 6.7.1.

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 from Section 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 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 12–3 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 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.

Table 12–3 shows the keywords for the `params` parameter.

Table 12–3 params Keywords for the SDO_JOIN Operator

Keyword	Description
mask	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. If this parameter is null or contains an empty string, `mask=FILTER` is assumed.
distance	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. If you specify `distance` but not `unit`, the unit of measurement associated with the data is assumed.
unit	Specifies a unit of measurement to be associated with the distance value (for example, `'distance=100 unit=meter'`). See Section 2.6 for more information about unit of measurement specification. If you specify `unit`, you must also specify `distance`. Data type is VARCHAR2. Default = unit of measurement associated with the data. For geodetic data, the default is meters.

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 a.name, b.name FROM cola_markets a, cola_markets b,
  TABLE(SDO_JOIN('COLA_MARKETS', 'SHAPE', 'COLA_MARKETS', 'SHAPE',
    'mask=ANYINTERACT')) c
```

```
  WHERE c.rowid1 = a.rowid AND c.rowid2 = b.rowid ORDER BY a.name;

NAME                             NAME
-------------------------------- --------------------------------
cola_a                           cola_a
cola_a                           cola_b
cola_a                           cola_c
cola_b                           cola_a
cola_b                           cola_b
cola_b                           cola_c
cola_c                           cola_a
cola_c                           cola_b
cola_c                           cola_c
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 12–4. 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 12–4 lists the keywords for the `param` parameter.

Table 12–4 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. Default = 1. For example: 'sdo_num_res=5'
unit	If the SDO_NN_DISTANCE ancillary operator is included in the call to SDO_NN, specifies the unit of measurement: a quoted string with `unit=` and an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table. See Section 2.6 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. For example: 'unit=KM'

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.

Specify the sdo_batch_size keyword if any geometries in the table might be nearer than the geometries specified in the WHERE clause. For example, 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;
```

If the sdo_batch_size keyword is not specified in this example, only the two nearest restaurants are returned, regardless of their CUISINE value; and if the CUISINE value of these two rows is not Italian, the query may return no rows. The ROWNUM <=2 clause is necessary to limit the number of results returned to no more than 2 where CUISINE is Italian.

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 4.1.6), 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 6.7.1.

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 there is an index associated with the column predicate in the WHERE clause, 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.)

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 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_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 C.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 from Section 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	Uses the `mask` keyword to specify the topological relationship of interest. This is a required parameter. Data type is VARCHAR2. 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. For backward compatibility, any additional keywords for the `param` 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.

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 the `geometry2` column is spatially indexed with the same SDO_LEVEL value as the `geometry1` column, the operator reuses the existing index, and performance is very good or better.
- If the `geometry2` column is spatially indexed with a different SDO_LEVEL value than the `geometry1` column, the operator reindexes `geometry2` in the same way as if there were no index on the column originally, and then 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 6.7.1.

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=inside+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. 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=inside+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=inside') = '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';
```

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 from Section 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 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 described 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`, `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.
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.

Value	Description
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 2.6 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

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 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 C.2.

Related Topics

- SDO_FILTER
- SDO_RELATE

13

Geometry Subprograms

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 13–1, and some usage information follows the table.

Table 13–1 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.

Table 13–1 (Cont.) Geometry Subprograms

Subprogram	Description
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.
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 2.4).

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 2.4).

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 12) 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 2.4).

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

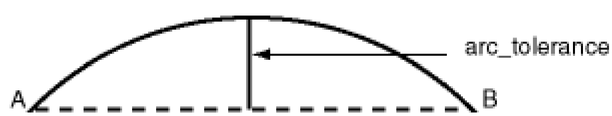
If you have geometries in a projected coordinate system that contain circles or circular arcs, you can use this function to densify them into regular polygons. You can then use the resulting straight-line polygon geometries for any Spatial operations, or you can transform them to any projected or geodetic coordinate system.

The `params` parameter is a quoted string that must contain the `arc_tolerance` keyword and that may contain 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 13–1 shows a line whose length is the `arc_tolerance` value for the arc between points A and B.

Figure 13–1 Arc Tolerance



The `arc_tolerance` keyword value must be greater than or equal to the tolerance value associated with the geometry. 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 2.6 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 2.4).

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 2.6 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 2.4).

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 with 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 2.6 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, this parameter is required, and `arc_tolerance` must be specified, because Spatial uses the 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 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 6.7.3.

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 6.8.

```
-- 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 2.4).

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.

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 6.7.3.

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 2.4).

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 6.7.3.

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 2.4).

geom2
Geometry object.

dim2

Dimensional information array corresponding to `geom2`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

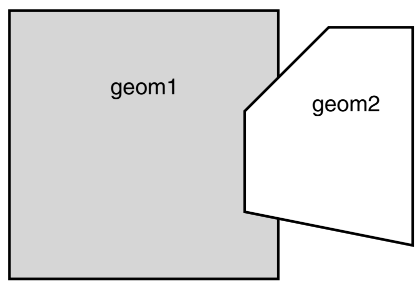
tol

Tolerance value (see Section 1.5.5).

Usage Notes

In Figure 13–2, the shaded area represents the polygon returned when SDO_DIFFERENCE is used with a square (`geom1`) and another polygon (`geom2`).

Figure 13–2 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 2.4).

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 2.4).

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 2.6 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 2.4).

geom2
Geometry object.

dim2

Dimensional information array corresponding to `geom2`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

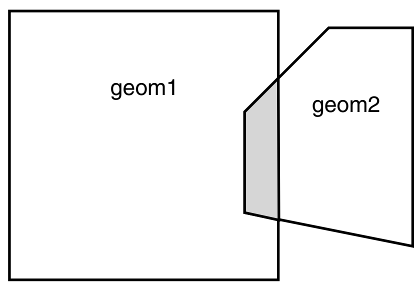
tol

Tolerance value (see Section 1.5.5).

Usage Notes

In Figure 13–3, the shaded area represents the polygon returned when SDO_INTERSECTION is used with a square (`geom1`) and another polygon (`geom2`).

Figure 13–3 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 2.4).

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 2.6 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 2.4).

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

This function is not supported with geodetic data.

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)
--------------------------------------------------
                                                10
```

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 2.4).

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.

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 2.4).

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

This function is not supported with geodetic data.

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)
--------------------------------------------------
                                                 6
```

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 2.4).

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 2.4).

geom2
Geometry object.

dim2

Dimensional information array corresponding to `geom2`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

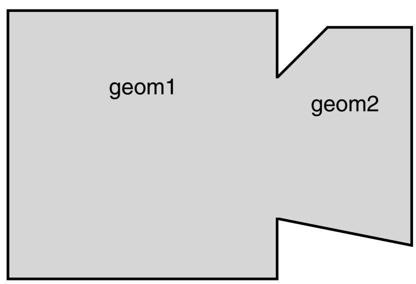
tol

Tolerance value (see Section 1.5.5).

Usage Notes

In Figure 13–4, the shaded area represents the polygon returned when SDO_UNION is used with a square (`geom1`) and another polygon (`geom2`).

Figure 13–4 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 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 19) 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 intersection 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 2.4).

geom2
Geometry object.

dim2

Dimensional information array corresponding to `geom2`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

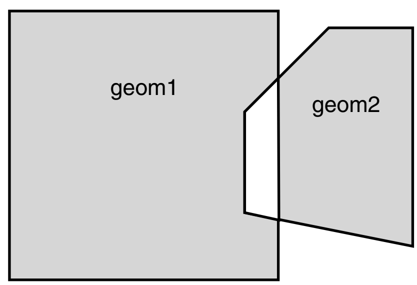
tol

Tolerance value (see Section 1.5.5).

Usage Notes

In Figure 13–5, the shaded area represents the polygon returned when SDO_XOR is used with a square (`geom1`) and another polygon (`geom2`).

Figure 13–5 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

Format

```
SDO_GEOM.VALIDATE_GEOMETRY(
    theGeometry  IN SDO_GEOMETRY,
    theDimInfo   IN SDO_DIM_ARRAY
    ) RETURN VARCHAR2;
```

or

```
SDO_GEOM.VALIDATE_GEOMETRY(
    theGeometry  IN SDO_GEOMETRY,
    tolerance    IN NUMBER
    ) RETURN VARCHAR2;
```

Description

Performs a consistency check for valid geometry types. The function checks the representation of the geometry from the tables against the element definitions.

Note: The VALIDATE_GEOMETRY function was deprecated in a previous release of Spatial. The current Spatial release is the last supported release for this function, and it will not be included in future releases of this guide. You should use instead the SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT function.

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 2.4).

tolerance
Tolerance value (see Section 1.5.5).

Usage Notes

This deprecated function performs the same checks as the SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT function; however, if the geometry is not valid, it does not return information about the context.

If the geometry is not valid, this function returns one of the following:

- An Oracle error message number based on the specific reason the geometry is invalid
- FALSE if the geometry fails for some other reason

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`.

Examples

The following example validates the geometry of `cola_c`. (The example uses the definitions and data from Section 2.1.)

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

NAME
--------------------------------
SDO_GEOM.VALIDATE_GEOMETRY(C.SHAPE,0.005)
--------------------------------------------------------------------------------
cola_c
TRUE
```

Related Topics

- SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
- SDO_GEOM.VALIDATE_LAYER

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 2.4).

tolerance
Tolerance value (see Section 1.5.5).

Usage Notes

If the geometry is valid, this function returns TRUE.

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)

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.

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>]
```

Related Topics

- SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT

SDO_GEOM.VALIDATE_LAYER

Format

```
SDO_GEOM.VALIDATE_LAYER(
    geom_table        IN VARCHAR2,
    geom_column       IN VARCHAR2,
    pkey_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.

> **Note:** The VALIDATE_LAYER procedure was deprecated in a previous release of Spatial. The current Spatial release is the last supported release for this procedure, and it will not be included in future releases of this guide. You should use instead the SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT procedure.

Parameters

geom_table
Spatial geometry table.

geom_column
Geometry object column to be examined.

pkey_column
The primary key column. This must be a single numeric (NUMBER data type) column.

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. If the primary key is indexed, you can look at the last PKEY_COLUMN value to see approximately how much of the validation is completed.

Usage Notes

This deprecated 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: (pkey_column NUMBER, result VARCHAR2(10)). 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 PKEY_COLUMN column contains the primary key value of the row containing the invalid geometry, and the RESULT column contains an Oracle error message number. You can then look up this error message to determine 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 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).
- 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).

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. (The example uses the definitions and data from Section 2.1.) In this case, a row with a RESULT value of DONE is written to the result table, because all the geometries are valid.

```
-- Is a layer valid? (First, create the result table.)
CREATE TABLE val_results (mkt_id number, result varchar2(10));
CALL SDO_GEOM.VALIDATE_LAYER('COLA_MARKETS','SHAPE','MKT_ID','VAL_RESULTS');

Call completed.

SELECT * from val_results;

    MKT_ID RESULT
---------- ----------
           DONE
```

Related Topics

- SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
- SDO_GEOM.VALIDATE_GEOMETRY

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).
- 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).

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>

AAABXNAABAAAK+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 2.4).

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 2.4).

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 2.6 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 12) 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

14 Spatial Aggregate Functions

This chapter contains reference and usage information for the spatial aggregate functions, which are listed in Table 14–1.

Table 14–1 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 geometry objects.
SDO_AGGR_UNION	Returns a geometry object that is the topological union (*OR* 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 13.

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.

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 19) 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 string 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 13.

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

Use this function instead of the deprecated SDO_TUNE.EXTENT_OF function to return the MBR of geometries. The SDO_TUNE.EXTENT_OF function is limited to two-dimensional geometries, whereas this function is not.

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.

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

See also the information about the SDO_GEOM.SDO_UNION function in Chapter 13.

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 C.4.

15

Coordinate System Transformation Subprograms

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 15–1 lists the coordinate system transformation subprograms.

Table 15–1 Subprograms for Coordinate System Transformation

Subprogram	Description
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.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.TRANSFORM

Format

```
SDO_CS.TRANSFORM(
    geom    IN SDO_GEOMETRY,
    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_srname IN VARCHAR2
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_CS.TRANSFORM(
    geom      IN SDO_GEOMETRY,
    dim       IN SDO_DIM_ARRAY,
    to_srname 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 MDSYS.CS_SRS table (described in Section 6.4.1).

dim
Dimensional information array corresponding to `geom`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

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 MDSYS.CS_SRS table (described in Section 6.4.1).

to_srname
The name of the coordinate system to be used for the transformation. It must be a value (specified exactly) in the CS_NAME column of the MDSYS.CS_SRS table (described in Section 6.4.1).

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_srname` 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 MDSYS.CS_SRS table (described in Section 6.4.1).

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 6.8.)

```
-- 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_srname 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);
```

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 MDSYS.CS_SRS table (described in Section 6.4.1).

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 15–2.

Table 15–2 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 *Oracle Database SQL Reference*.
GEOMETRY	SDO_GEOMETRY	Geometry object with coordinate values in the specified (`to_srid` 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 6.8.)

```
-- 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 6–5 in Section 6.8 includes a display of the geometry object coordinates in both tables (COLA_MARKETS_CS and COLA_MARKETS_CS_8199).

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 MDSYS.CS_SRS table (described in Section 6.4.1).

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 6.4.1.1.

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 MDSYS.CS_SRS table (described in Section 6.4.1) 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 6.8.)

```
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
```

16

Linear Referencing Subprograms

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.OFFSET_GEOM_SEGMENT	Returns the geometric segment at a specified offset from a geometric segment.
SDO_LRS.SCALE_GEOM_SEGMENT (deprecated)	Scales a geometric segment.
SDO_LRS.SPLIT_GEOM_SEGMENT	Splits a geometric segment into two segments.

Table 16–1 (Cont.) Subprograms for Creating and Editing Geometric Segments

Subprogram	Description
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.
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.

Table 16–2 (Cont.) Subprograms for Querying and Validating Geometric Segments

Subprogram	Description
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).
SDO_LRS.IS_SHAPE_PT_MEASURE	Checks if a specified measure value is 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.

Table 16–2 (Cont.) Subprograms for Querying and Validating Geometric Segments

Subprogram	Description
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.
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
    ) 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 2.4).

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).

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
    ) 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 2.4).

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 2.4).

tolerance

Tolerance value (see Section 1.5.5 and Section 7.6).

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
    ) 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 2.4).

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 2.4).

tolerance
Tolerance value (see Section 1.5.5 and Section 7.6).

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 2.4).

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).

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 2.4).

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).

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, .00
5), 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 2.4).

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 `diminfo`.

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 2.4).

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 2.4).

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
    ) 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 2.4).

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).

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 2.4).

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
    ) 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).

dim_array
Dimensional information array corresponding to `geom_segment`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

point_dim_array

Dimensional information array corresponding to `point`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

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 2.4).

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 2.4).

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 2.4).

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 2.4).

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 2.4).

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 2.4).

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
--------------------------------------------------------------------------------
                                                                               9
```

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 2.4).

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 after 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 2.4).

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 2.4).

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 2.4).

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 2.4).

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 2.4).

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 2.4).

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 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 2.4).

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 2.4).

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.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 2.4).

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 2.4).

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
    [, 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 2.4).

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).

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 2.6 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 13 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. (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))

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.
```

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 2.4).

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 this 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
  HERE 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
    [, 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 2.4).

point
Point to be projected.

tolerance
Tolerance value (see Section 1.5.5 and Section 7.6).

point_dim_array
Dimensional information array corresponding to `point`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

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 2.4).

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 2.4).

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_segment`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

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 19.

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_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_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 2.4).

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 between release 8.1.7 and the current release. 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 the current release, 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.SCALE_GEOM_SEGMENT

Format

```
SDO_LRS.SCALE_GEOM_SEGMENT(
    geom_segment  IN SDO_GEOMETRY,
    start_measure IN NUMBER,
    end_measure   IN NUMBER,
    shift_measure IN NUMBER
    ) RETURN SDO_GEOMETRY;
```

or

```
SDO_LRS.SCALE_GEOM_SEGMENT(
    geom_segment  IN SDO_GEOMETRY,
    dim_array     IN SDO_DIM_ARRAY,
    start_measure IN NUMBER,
    end_measure   IN NUMBER,
    shift_measure IN NUMBER
    ) RETURN SDO_GEOMETRY;
```

Description

Returns the geometry object resulting from the scaling of a geometric segment.

Parameters

geom_segment
Geometric segment to be scaled.

dim_array
Dimensional information array corresponding to `geom_segment`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

start_measure
Start measure of the scaled geometric segment.

end_measure
End measure of the scaled geometric segment.

shift_measure
Shift measure of the scaled geometric segment.

Usage Notes

This function performs a general scaling operation to the geometric segment. The new start and end measures are assigned, and all measures are populated by a linear mapping between old and new start and end measures. The shift measure is applied to the segment after scaling.

> **Note:** This general-purpose function was deprecated in a previous release of Spatial. The current Spatial release is the last supported release for this function, and it will not be included in future releases of this guide. You should instead use other functions for specific purposes, as described in Table 16–4.

Table 16–4 lists some common tasks and the suggested functions to use instead of SCALE_GEOM_SEGMENT.

Table 16–4 Functions to Use Instead of SCALE_GEOM_SEGMENT

Task	Suggested Function
Shift all measures by a specified amount (for example, to accommodate new construction at the start of a road that causes the original start point to be *n* measure units beyond the new start point).	SDO_LRS.TRANSLATE_MEASURE
Reverse the direction of a segment (for example, to allow one road segment to be concatenated with another coming from the opposite direction, because both segments to be concatenated must have the same direction).	SDO_LRS.REVERSE_GEOMETRY
Scale the measure information without performing a shift (for example, to change the measures from miles to kilometers).	SDO_LRS.REDEFINE_GEOM_SEGMENT

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.

For more information about scaling a geometric segment, see Section 7.5.6.

Examples

The following examples illustrate some SCALE_GEOM_ELEMENT uses. (These examples use the definitions from the example in Section 7.7.)

```
-- Shift by 5 (for example, 5-mile segment added before original start)
SELECT  SDO_LRS.SCALE_GEOM_SEGMENT(a.route_geometry, m.diminfo, 0, 27, 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.SCALE_GEOM_SEGMENT(A.ROUTE_GEOMETRY,M.DIMINFO,0,27,5)(SDO_GTYPE, SDO_SRI
--------------------------------------------------------------------------------
SDO_GEOMETRY(3002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
2, 2, 5, 2, 4, 7, 8, 4, 13, 12, 4, 17, 12, 10, 23, 8, 10, 27, 5, 14, 32))

-- "Convert" mile measures to kilometers (27 * 1.609 = 43.443)
SELECT  SDO_LRS.SCALE_GEOM_SEGMENT(route_geometry, 0, 43.443, 0)
  FROM lrs_routes WHERE route_id = 1;

SDO_LRS.SCALE_GEOM_SEGMENT(ROUTE_GEOMETRY,0,43.443,0)(SDO_GTYPE, SDO_SRID, SDO_P
--------------------------------------------------------------------------------
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.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 2.4).

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 2.4).

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 2.4).

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.

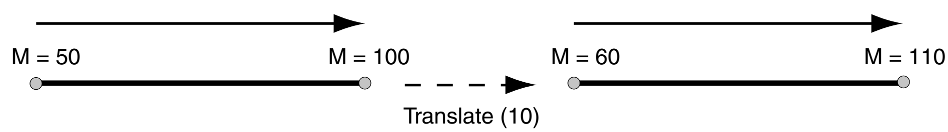
dim_array
Dimensional information array corresponding to `geom_segment`, usually selected from one of the xxx_SDO_GEOM_METADATA views (described in Section 2.4).

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 2.4).

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 2.4).

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 2.4).

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 2.4).

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
```

17

SDO_MIGRATE Procedure

The SDO_MIGRATE.TO_CURRENT procedure described in this chapter lets you upgrade spatial geometry tables from previous releases of Spatial.

SDO_MIGRATE.TO_CURRENT is the only procedure that you should use for upgrading. 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 (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. The format depends on whether you are upgrading from the Spatial relational model (release 8.1.5 or lower) or object-relational model (release 8.1.6 or higher). See the Usage Notes for the model that applies to you.

You should use this procedure for any spatial data 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

tabname
Table with geometry objects.

column_name
Column in `tabname` 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.

layer
Name of the layer to be upgraded.

newtabname
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 Upgrade

All geometry objects in `tabname` will be 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.

The procedure also orders geometries so that exterior rings are followed by their interior rings, and saves them 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 calling this procedure.
- 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 saves them 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 higher format to the format of the current release.

```
SQL> execute sdo_migrate.to_current('ROADS');
```

18

Spatial Tuning Subprograms

This chapter contains descriptions of the tuning subprograms shown in Table 18–1.

Table 18–1 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 (deprecated)	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.

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

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 4.1.2 explains geodetic indexes.)

Usage Notes

The function returns the estimated maximum number of megabytes needed for the spatial index table (described in Section 2.5.2) 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, tabname, 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 4.1.2).

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.

> **Note:** This function is deprecated, and will not be supported in future versions of Spatial. You are instead encouraged to use the SDO_AGGR_MBR function, documented in Chapter 14, to return the MBR of geometries. The SDO_TUNE.EXTENT_OF function is limited to two-dimensional geometries, whereas the SDO_AGGR_MBR function is not.

Parameters

table_name
Spatial geometry table.

column_name
Geometry column for which the minimum bounding rectangle is to be returned.

Usage Notes

This deprecated function returns NULL if the data is inconsistent.

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 14)

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

Provides information about each geometry type stored in a column of type SDO_GEOMETRY.

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;
```

Description

Returns the quality degradation for an index or the average quality degradation for all index tables for an index.

Parameters

schemaname
Name of the schema that contains the index specified in `indexname`.

indexname
Name of the spatial R-tree index.

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.

Index names are available through the xxx_SDO_INDEX_INFO and xxx_SDO_INDEX_METADATA views, which are described in Section 2.5.1.

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
```

19

Spatial Utility Subprograms

This chapter contains descriptions of the spatial utility subprograms shown in Table 19–1.

Table 19–1 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.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.

Table 19–1 (Cont.) Spatial Utility Subprograms

Subprogram	Description
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.REMOVE_DUPLICATE_VERTICES	Removes duplicate (redundant) vertices from a geometry.
SDO_UTIL.REVERSE_LINESTRING	Returns a line string geometry with the vertices of the input geometry in reverse order.
SDO_UTIL.SIMPLIFY	Simplifies the input geometry, based on a threshold value, using the Douglas-Peucker algorithm.
SDO_UTIL.TO_GMLGEOMETRY	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.

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 13. 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 13)

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 13.)

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 circle radius 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.

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 14) 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 14)
- 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.ANGLE_UNITS table (described in Section 6.4.2), the MDSYS.SDO_AREA_UNITS table (described in Section 2.6), or the MDSYS.SDO_DIST_UNITS table (described in Section 2.6). 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 *pi*). 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 13.)

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_TYPE 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 2.3.2, 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 13), 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 2.3.6.

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 2–3 in Section 2.3.2), 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.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 13), 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 10*g* Release 1 (10.1), the Oracle transportable tablespace feature could not be used with tablespaces that contained any spatial indexes. Effective with Oracle Database 10*g* 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.

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.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-DIMENSIONALPOLYGONNUL
--------------------------------------------------------------------------------
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
    ) 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.

Usage Notes

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 then 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, individual elements that did not overlap before simplification might now overlap. If overlapping of elements occurs in a multipolygon, the geometry is invalid because Open GIS Consortium rules state that polygon elements may not overlap in a multipolygon. Under certain conditions, single polygon geometries might also become invalid.

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.

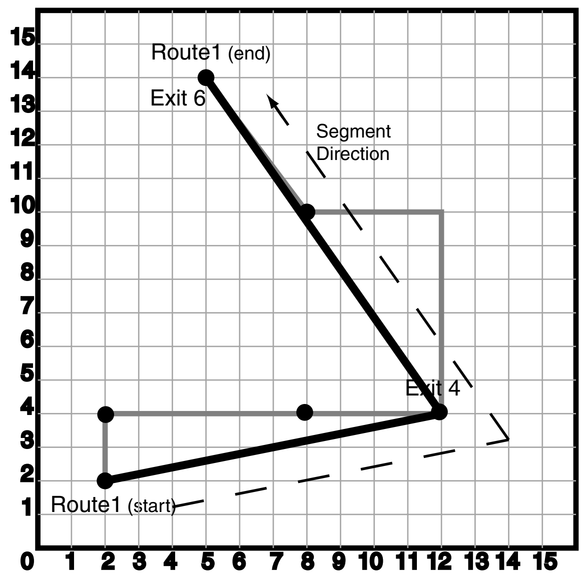
```
SELECT SDO_UTIL.SIMPLIFY(
  SDO_GEOMETRY(
    3302,  -- line string, 3 dimensions (X,Y,M), 3rd is linear ref. dimension
    NULL,
```

```
    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
) 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 19–1 shows the result of this example. In Figure 19–1, the thick solid black line is the resulting geometry, and the thin solid light line between the start and end points is the input geometry.

Figure 19–1 Simplification of a Geometry



Related Topics

None.

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 Oracle9*i* and higher 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_BUFFER or SDO_GEOM.SDO_ARC_DENSIFY 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:outerBou
ndaryIs><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:outerBou
ndaryIs><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:ou
terBoundaryIs><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:LinearRin
g></gml:outerBoundaryIs><gml:innerBoundaryIs><gml:LinearRing><gml:coordinates de
cimal="." cs="," ts=" ">30.3,30.3 30.3,40.1 40.1,40.1 30.3,30.3 </gml:coordinate
s></gml:LinearRing></gml:innerBoundaryIs><gml:innerBoundaryIs><gml:LinearRing><g
ml: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:ou
terBoundaryIs><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 decima
l="." cs="," ts=" ">30.3,30.3 30.3,40.1 40.1,40.1 40.1,30.3 30.3,30.3 </gml:coor
dinates></gml:LinearRing></gml:innerBoundaryIs><gml:innerBoundaryIs><gml:LinearR
ing><gml:coordinates decimal="." cs="," ts=" ">5,5 5,6 6,6 6,5 5,5 </gml:coordin
ates></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');

THEXMLELEMENTS
--------------------------------------------------------------------------------
<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.

20

Geocoding Subprograms

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 20–1 lists the geocoding subprograms.

Table 20–1 Subprograms for Geocoding Address Data

Subprogram	Description
SDO_GCDR.GEOCODE	Geocodes an unformatted address and returns an SDO_GEOR_ADDR 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.

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_GEOR_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('1CARLTONBGOODLETTPL','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_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_GEOR_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_GEOR_ADDR 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, using the `RELAX_BASE_NAME` match mode. 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', 'RELAX_BASE_NAME') FROM DUAL;

SDO_GCDR.GEOCODE_AS_GEOMETRY('SCOTT',SDO_KEYWORDARRAY('1CARLTONBGOODLETTPL','
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.41815, 37.7784183, NULL), NULL, NUL
L)
```

21

Spatial Analysis and Mining Subprograms

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 21–1 lists the spatial analysis and mining subprograms.

Table 21–1 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.

Table 21–1 (Cont.) Subprograms for Spatial Analysis and Mining

Function	Description
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 containing a distance value and optionally a unit value. 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 the distance keyword and that may contain the unit keyword to identify the unit of measurement associated with the distance value. For example:

```
'distance=2 unit=km'
```

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 2.6 for more information about unit of measurement specification.

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.aggregates_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 containing a distance value and optionally a unit value. See the Usage Notes 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.

The `dst_spec` parameter, if specified, is a quoted string that must contain the `distance` keyword and that may contain the `unit` keyword to identify the unit of measurement associated with the `distance` value. For example:

```
'distance=2 unit=km'
```

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 2.6 for more information about unit of measurement specification.

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 bin geometries.

bin_tablename
Name of the table that holds the bin geometries.

bin_colname

Column in `bin_tablename` that holds the bin geometries.

Usage Notes

This function bins the geometry to the most-intersecting bin in the specified bin table.

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 `table_name` 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 bin geometries.

bin_id_colname
Name of the column in `bin_tablename` that holds the bin ID value of each geometry added to a bin.

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`.

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. 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 that must contain the `distance` keyword and that may contain the `unit` keyword to identify the unit of measurement associated with the `distance` value. For example:

```
'distance=2 unit=km'
```

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 2.6 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 19) 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 function.

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,
    tiling_domain    IN SDO_DIM_ARRAY 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.

tiling_domain
Domain for the tiling level. If the geometry data in the `theme_name` table is geodetic, you must specify this parameter. If the geometry data in the `theme_name` table is not geodetic and if you do not specify this parameter, the extent associated with the `theme_name` table is used.

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, 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.

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_ARR
AY(-180, -45, -90, 0))
     .034831005

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(-180, 0, -90, 45))
     7.73307783

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(-90, -90, 0, -45))
     .019498368

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(-90, -45, 0, 0))
     .939061456

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(-90, 0, 0, 45))
     1.26691592

SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3), SDO_ORDINATE_ARR
AY(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,
    srid         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.

srid
SRID value to be included for the coordinate system in the returned tile geometries.

Usage Notes

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.
```

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 methods, operators, functions, and procedures.
- Part III provides supplementary information (appendixes and a glossary).

Part III contains the following:

A

Installation, Compatibility, and Upgrade

If you are upgrading to Oracle Database 10*g*, Oracle Spatial is automatically upgraded as part of the operation. For information about the upgrade procedure, see *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.

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 higher, 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 higher (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;
```

B

Oracle Locator

Oracle Locator (also referred to as Locator) is a feature of Oracle Database 10*g* 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 10*g*. Spatial is a priced option available only with Oracle Database 10*g* Enterprise Edition. Spatial includes all Locator features as well as other features that are not available with Locator.

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 12) that use the spatial index for performing spatial queries
- Some geometry functions and the SDO_AGGR_MBR spatial aggregate function
- Integration with Oracle Application Server 10*g* Wireless

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 4.1.6 and Section 4.1.7
Geodetic data support	Section 6.2 and Section 6.4
SQL statements for creating, altering, and deleting indexes (except deferred updates to spatial indexes, as noted in Table B–2)	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	Chapter 11
Spatial operators	Chapter 12
Implicit coordinate system transformations for operator calls where a window needs to be converted to the coordinate system of the queried layer	Chapter 12
The following SDO_GEOM package functions and procedures: SDO_GEOM.SDO_DISTANCE SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT SDO_GEOM.VALIDATE_GEOMETRY (deprecated) SDO_GEOM.VALIDATE_LAYER (deprecated)	Chapter 13
SDO_AGGR_MBR spatial aggregate function (new to Locator with release 9.2)	Chapter 14
Package (SDO_MIGRATE) to upgrade data from previous Spatial releases to the current release	Chapter 17
Object replication	*Oracle Database Advanced Replication*
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
Deferred updates to spatial indexes (`'index_status=deferred'` with the ALTER INDEX statement)	Chapter 10
SDO_GEOM package functions and procedures, except for those listed in Table B–1	Chapter 13
Spatial aggregate functions, except for any listed in Table B–1	Chapter 14
Linear referencing system (LRS) support	Chapter 7 (concepts and usage) and Chapter 16 (reference)
Coordinate system support for explicit geometry and layer transformations (SDO_CS.TRANSFORM function and SDO_CS.TRANSFORM_LAYER procedure)	Chapter 15
Tuning functions and procedures (SDO_TUNE package)	Chapter 18
Spatial utility functions (SDO_UTIL package)	Chapter 19
Spatial analysis and mining functions and procedures (SDO_SAM package)	Chapter 21
Geocoding support (SDO_GCDR package)	Chapter 5 (concepts and usage) and Chapter 20 (reference)
GeoRaster support	*Oracle Spatial GeoRaster*
Topology data model	*Oracle Spatial Topology and Network Data Models*
Network data model	*Oracle Spatial Topology and Network Data Models*

Although Locator is available on both the Standard and Enterprise Editions of Oracle Database 10*g*, 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 *Oracle Database Advanced Replication* for more information.)
Partitioned spatial indexes	Requires the Partitioning Option with Enterprise Edition. Not supported with Standard Edition.

C

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:

C.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);
```

C.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 C–1 finds all cities within 15 miles of the interstate highway I170.

Example C–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 C–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`): the LOCATION column of the GEOD_CITIES table.
- `i.geom` specifies the query window (`aGeom`): the spatial geometry in the GEOM column of the GEOD_INTERSTATES table, in the row whose HIGHWAY column contains the value `I170`.

Example C–2 finds all interstate highways within 15 miles of the city of Tampa.

Example C–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 C–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`): the GEOM column of the GEOD_INTERSTATES table.
- `c.location` specifies the query window (`aGeom`): the spatial geometry in the LOCATION column of the GEOD_CITIES table, in the row whose CITY column contains the value `Tampa`.

C.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. 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 `sdo_batch_size=5` will return no rows because the `sdo_num_res` keyword only takes proximity into account, and not any conditions in the WHERE clause.

If you specify the optional `sdo_batch_size` 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 C–3 finds the five cities nearest to the interstate highway I170 and the distance in miles for each city, ordered by distance in miles.

Example C–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 C–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 INTERSTATES table, and `I170` is a value from the HIGHWAY column.
- `c.location` specifies the search column (`geometry1`): the LOCATION column of the GEOD_CITIES table.
- `i.geom` specifies the query window (`geometry2`): 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 to the call to SDO_NN_DISTANCE.

In Example C–3, `ORDER BY distance_in_miles` orders the results from the WHERE clause by distance in miles.

The statement in Example C–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 C–4 extends Example C–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, and the distance in miles for each city, ordered by distance in miles.

Example C–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 C–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 C–4, `ORDER BY distance_in_miles` orders the results from the WHERE clause by distance in miles.

The statement in Example C–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
```

C.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 C–5 aggregates all the counties in Texas, producing the boundary for the state of Texas.

Example C–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.

authalic sphere

A sphere that has the same surface area as a particular oblate ellipsoid of revolution representing the figure of the Earth.

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.

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 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 the two objects' geometries 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.

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.

keyword

Synonym for reserved word.

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.

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.

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.

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.

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.

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 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 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 and/or location 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.

Index

Numerics

A

B

C

F

G

H

I

J

L

M

N

O

P

Q

R

S

W

X

Z