Oracle® Spatial

Topology and Network Data Models

10*g* Release 1 (10.1)

Part No. B10828-01

December 2003

Provides usage and reference information about the topology data model and network data model capabilities of Oracle Spatial.

Oracle Spatial Topology and Network Data Models, 10*g* Release 1 (10.1)

Part No. B10828-01

Primary Author: Chuck Murray

Contributors: Bruce Blackwell, Janet Blowney, Frank Lee, Siva Ravada, Jack Wang

Contents

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Spatial Topology and Network Data Models, 10*g* Release 1 (10.1)

Part No. B10828-01

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

Oracle Spatial Topology and Network Data Models provides usage and reference information about the topology data model and network data model of Oracle Spatial, which is often referred to as just Spatial.

This preface contains these topics:

Audience

This guide is intended for those who need to use the Spatial topology or network data model to work with data about nodes, edges, and faces in a topology or nodes, links, and paths in a network.

You are assumed to be familiar with the main Spatial concepts, data types, and operations, as documented in *Oracle Spatial User's Guide and Reference*.

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. The first part contains conceptual, usage, and reference information about the topology data model of Oracle Spatial. The second part contains conceptual, usage, and reference information about the network data model of Oracle Spatial. If you develop network applications, you must understand the main concepts in both parts.

This guide has the following elements.

Part I, "Topology Data Model"

Contains chapters with conceptual, usage, and reference information about the topology data model of Oracle Spatial.

Chapter 1, "Topology Data Model Overview"

Provides conceptual and usage information about the topology data model.

Chapter 2, "Editing Topologies"

Explains how to edit node and edge data in a topology. The operations include adding, moving, and removing nodes and edges.

Chapter 3, "SDO_TOPO Package Subprograms"

Provides reference information about procedures in the PL/SQL package MDSYS.SDO_TOPO.

Chapter 4, "SDO_TOPO_MAP Package Subprograms"

Provides reference information about procedures in the PL/SQL package MDSYS.SDO_TOPO_MAP.

Chapter 5, "Topology Operators"

Provides reference information about topology operators (SDO_ANYINTERACT in the current release).

Part II, "Network Data Model"

Contains chapters with conceptual, usage, and reference information about the network data model of Oracle Spatial.

Chapter 6, "Network Data Model Overview"

Provides conceptual and usage information about the network data model.

Chapter 7, "SDO_NET Package Subprograms"

Provides reference information about procedures in the PL/SQL package MDSYS.SDO_NET.

Related Documentation

For more information, see *Oracle Spatial User's Guide and Reference*.

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

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

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.
`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.

Part I

Topology Data Model

This document has two main parts:

- Part I provides conceptual, usage, and reference information about the topology data model of Oracle Spatial.
- Part II provides conceptual, usage, and reference information about the network data model of Oracle Spatial.

Much of the conceptual information in Part I also applies to the network data model. Therefore, if you develop network applications, you should be familiar with the main terms and concepts from both parts of this document.

Part I contains the following chapters:

1 Topology Data Model Overview

The topology data model of Oracle Spatial lets you work with data about nodes, edges, and faces in a topology. For example, United States Census geographic data is provided in terms of nodes, chains, and polygons, and this data can be represented using the Spatial topology data model. You can store information about topological elements and geometry layers in Oracle Spatial tables and metadata views. You can then 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).

This chapter describes the Spatial data structures and data types that support the topology data model, and what you need to do to populate and manipulate the structures. You can use this information to write a program to convert your topological data into formats usable with Spatial.

A demo procedure is provided that processes U.S. Census topological data for use with Spatial, although you must modify that procedure (or write your own) to process your own topological data for use with Spatial. For information about the demo files provided, see the files in the `demos` directory and its subdirectory hierarchy under your Spatial Topology Manager installation directory. For information about the spatial topology editor demo, see `demos/Topology/Bulk-Load/README`.

Note: Although this chapter discusses some topology terms as they relate to Oracle Spatial, it assumes that you are familiar with basic topology concepts.

It also assumes that you are familiar with the main Spatial concepts, data types, and operations, as documented in *Oracle Spatial User's Guide and Reference*.

This chapter contains the following major sections:

1.1 Main Steps in Using Topology Data

This section summarizes the main steps for working with topological data in Oracle Spatial. It refers to important concepts, structures, and operations that are described in detail in other sections.

The main steps for working with topological data are as follows:

1. Create the topology, using the SDO_TOPO.CREATE_TOPOLOGY procedure. This causes the <topology-name>_EDGE$, <topology-name>_NODE$, <topology-name>_FACE$, and <topology-name>_HISTORY$ tables to be created. (These tables are described in Section 1.5.1, Section 1.5.2, Section 1.5.3, and Section 1.5.5, respectively.)
2. Load topology data into the node, edge, and face tables created in Step 1. This is typically done using a bulk-load utility, but it can be done using SQL INSERT statements.
3. Create a feature table for each feature in the topology. For example, a city data topology might have separate feature tables for land parcels, streets, and traffic signs.
4. Associate the feature tables with the topology, using the SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER procedure for each feature table. This causes the

<topology-name>_RELATION$ table to be created. (This table is described in Section 1.5.4.)

5. Initialize topology metadata, using the SDO_TOPO.INITIALIZE_METADATA procedure. (This procedure also creates spatial indexes on the <topology-name>_EDGE$, <topology-name>_NODE$, and <topology-name>_FACE$ tables, and B-tree indexes on the <topology-name>_EDGE$, <topology-name>_RELATION$, and <topology-name>_HISTORY$ tables.)
6. Load the feature tables using the SDO_TOPO_GEOMETRY constructor. (This constructor is described in Section 1.6.2.)
7. Query the topology data (for example, using SDO_ANYINTERACT operator).
8. Optionally, edit topology data using the PL/SQL or Java application programming interfaces (APIs).

Section 1.11 contains a PL/SQL example that performs these main steps.

You can use the topology data model PL/SQL and Java APIs to update the topology (for example, to change the data about an edge, node, or face). The PL/SQL API for most editing operations is the SDO_TOPO_MAP package, which is documented in Chapter 4. The Java API is described in Section 1.8.1.

1.2 Topology Data Model Concepts

Topology is a branch of mathematics concerned with objects in space. Topological relationships include such relationships as *contains*, *inside*, *covers*, *covered by*, *touch*, and *overlap with boundaries intersecting*. Topological relationships remain constant when the coordinate space is deformed, such as by twisting or stretching. (Examples of relationships that are not topological include *length of*, *distance between*, and *area of*.)

The basic elements in a topology are its nodes, edges, and faces.

A **node**, represented by a point, can be isolated or it can be used to bound edges. Two or more edges meet at every non-isolated node. A node has a coordinate pair associated with it that describes the spatial location for that node. Examples of geographic entities that might be represented as nodes include start and end points of streets, places of historical interest, and airports (if the map scale is sufficiently large).

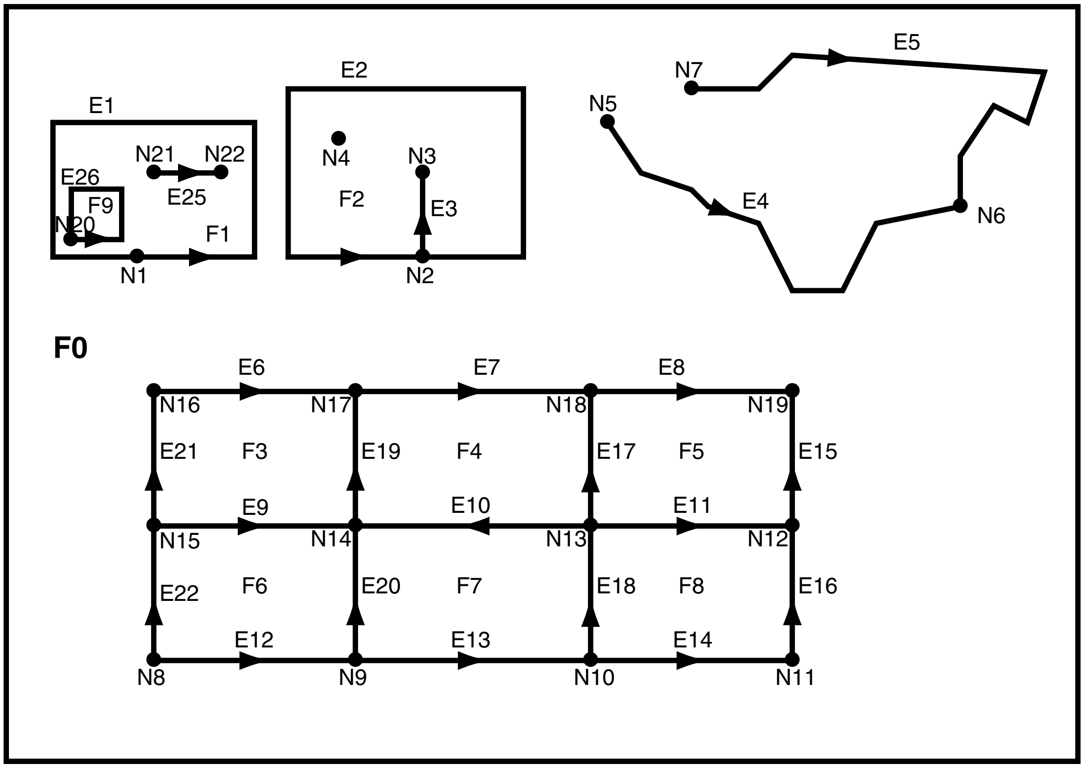
An **edge** is bounded by two nodes: the start (origin) node and the end (terminal) node. An edge has an associated geometric object, usually a coordinate string that describes the spatial representation of the edge. An edge may have several vertices making up a line string, circular arc string, or combination. Examples of geographic entities that might be represented as edges include segments of streets and rivers.

The order of the coordinates gives a **direction** to an edge, and direction is important in determining topological relationships. The positive direction agrees with the orientation of the underlying edge, and the negative direction reverses this orientation. Each orientation of an edge is referred to as a directed edge, and each directed edge is the mirror image of its other directed edge. The start node of the positive directed edge is the end node of the negative directed edge. An edge also lies between two faces and has references to both of them. Each directed edge contains a reference to the next edge in the contiguous perimeter of the face on its left side.

A **face**, represented by a polygon, has a reference to one directed edge of its outer boundary. If any island nodes or island edges are present, it also has a reference to one directed edge on the boundary of each island. Examples of geographic entities that might be represented as faces include parks, lakes, counties, and states.

Figure 1–1 shows a simplified topology containing nodes, edges, and faces. The arrowheads on each edge indicate the positive direction of the edge (or, more precisely, the orientation of the underlying line string or curve geometry for positive direction of the edge).

Figure 1–1 Simplified Topology



Notes on Figure 1–1:

- *E* elements (E1, E2, and so on) are edges, *F* elements (F0, F1, and so on) are faces, and *N* elements (N1, N2, and so on) are nodes.
- **F0** (face zero) is created for every topology. It is the universal face containing everything else in the topology. There is no geometry associated with the universal face. F0 has the face ID value of -1 (negative 1).
- There is a node created for every point geometry and for every start and end node of an edge. For example, face F1 has only one edge (a closed edge), E1. The edge has the same node as the start and end nodes (N1).
- An **island node** is a node that is isolated in a face. For example, node N4 is an island node in face F2.

- An **island edge** is an edge that is isolated in a face. For example, edge E25 is an island edge in face F1.
- An edge cannot have an island node on it. The edge can broken up into two edges by adding a node on the edge. For example, if there was originally a single edge between nodes N16 and N18, adding node N17 resulted in two edges: E6 and E7.
- Information about the topological relationships is stored in special edge, face, and node information tables. For example, the edge information table contains the following information about edges E9 and E10. (Note the direction of the arrowheads for each edge.) The next and previous edges are based on the left and right faces of the edge.

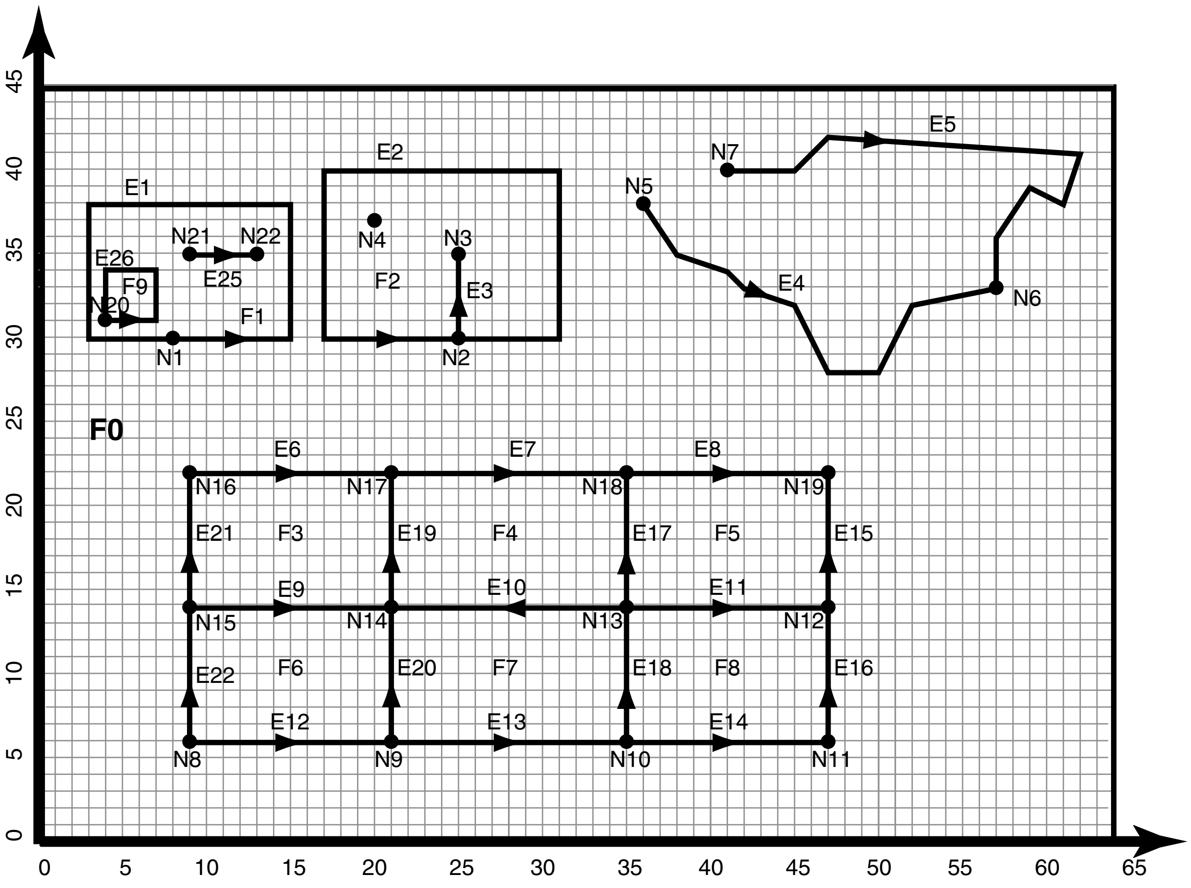
  For edge E9, the start node is N15 and the end node is N14, the next left edge is E19 and the previous left edge is -E21, the next right edge is -E22 and the previous right edge is E20, the left face is F3 and the right face is F6.

  For edge E10, the start node is N13 and the end node is N14, the next left edge is -E20 and the previous left edge is E18, the next right edge is E17 and the previous right edge is -E19, the left face is F7 and the right face is F4.

  For additional examples of edge-related data, including an illustration and explanations, see Section 1.5.1.

Figure 1–2 shows the same topology illustrated in Figure 1–1, but it adds a grid and unit numbers along the x-axis and y-axis. Figure 1–2 is useful for understanding the output of some of the examples in Chapter 3 and Chapter 4.

Figure 1–2 Simplified Topology, with Grid Lines and Unit Numbers



1.3 Topology Geometries and Layers

A **topology geometry** (also referred to as a **feature**) is a spatial representation of a real world object. For example, *Main Street* and *Walden State Park* might be the names of topology geometries. The geometry is stored as a set of topological elements (nodes, edges, and faces). Each topology geometry has a unique ID (assigned by Spatial when records are imported or loaded) associated with it.

A **topology geometry layer** is the collection of topology geometries of a specific type. For example, *Streets* might be the topology geometry layer that includes the

Main Street topology geometry, and *State Parks* might be the topology geometry layer that includes the *Walden State Park* topology geometry. Each topology geometry layer has a unique ID (assigned by Spatial) associated with it. The data for each topology geometry layer is stored in a **feature table**. For example, a feature table named CITY_STREETS might contain information about all topology geometries (individual roads or streets) in the *Streets* topology geometry layer.

Each topology geometry (feature) is defined as an object of type SDO_TOPO_GEOMETRY (described in Section 1.6.1), which identifies the topology geometry type, topology geometry ID, topology geometry layer ID, and topology ID for the topology.

Topology metadata is automatically maintained by Spatial in the USER_SDO_TOPO_METADATA and ALL_SDO_TOPO_METADATA views, which are described in Section 1.7.2. The USER_SDO_TOPO_INFO and ALL_SDO_TOPO_INFO views (described in Section 1.7.1) contain a subset of this topology metadata.

1.3.1 Features and Topology Objects

Often, there are fewer features in a topology than there are nodes, edges, and faces. For example, a road feature may consist of many edges, an area feature such as a park may consist of many faces, and some nodes may not be associated with point features. Figure 1–3 shows point, line, and area features associated with the topology that was shown in Figure 1–1 in Section 1.2.

Figure 1–3 Features in a Topology

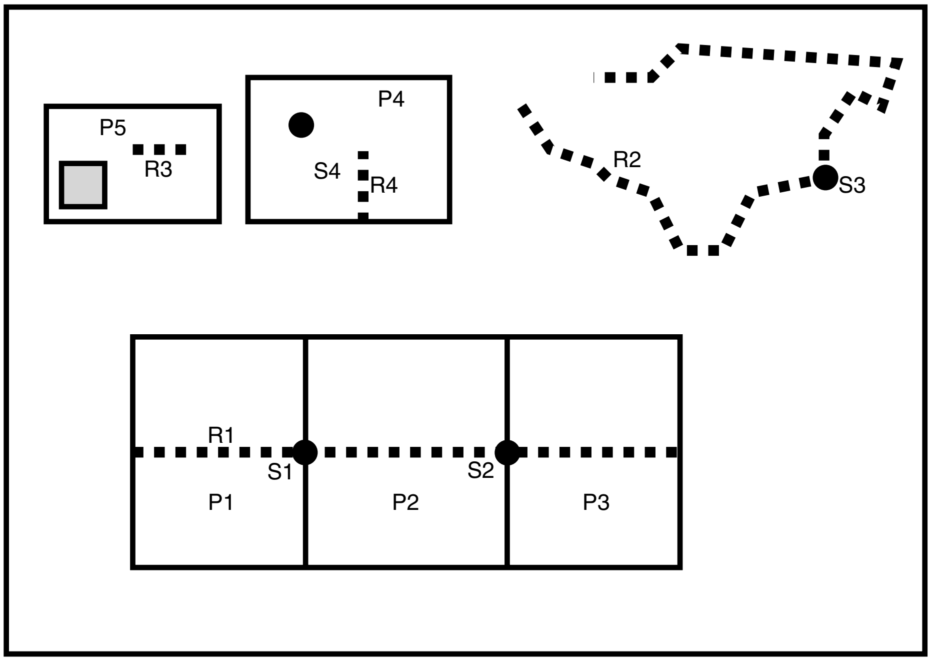


Figure 1–3 shows the following kinds of features in the topology:

- Point features (traffic signs), shown as dark circles: `S1`, `S2`, `S3`, and `S4`
- Linear features (roads or streets), shown as dashed lines: `R1`, `R2`, `R3`, and `R4`
- Area features (land parcels), shown as rectangles: `P1`, `P2`, `P3`, `P4`, and `P5`

  Land parcel `P5` does not include the shaded area within its area. (Specifically, `P5` includes face `F1` but not face `F9`. These faces are shown in Figure 1–1 in Section 1.2.)

Example 1–8 in Section 1.11 defines these features.

1.4 Topology Geometry Layer Hierarchy

In some topologies, the topology geometry layers (feature layers) have one or more parent-child relationships in a **topology hierarchy**. That is, the layer at the topmost level consists of features in its child layer at the next level down in the hierarchy; the child layer might consist of features in its child layer at the next layer farther down; and so on. For example, a land use topology might have the following topology geometry layers at different levels of hierarchy:

- States at the highest level, which consists of features from its child layer, Counties
- Counties at the next level down, which consists of features from its child layer, Tracts
- Tracts at the next level down, which consists of features from its child layer, Block Groups
- Block Groups at the next level down, which consists of features from its child layer, Land Parcels
- Land Parcels at the lowest level of the hierarchy

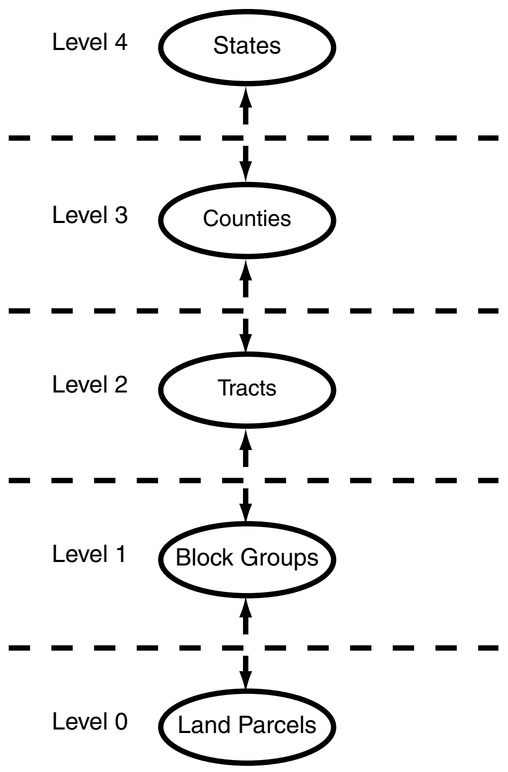
If the topology geometry layers in a topology have this hierarchical relationship, it is far more efficient if you model the layers as hierarchical than if you specify all topology geometry layers at a single level (that is, with no hierarchy). For example, it is more efficient to construct SDO_TOPO_GEOMETRY objects for counties by specifying only the tracts in the county than by specifying all land parcels in all block groups in all tracts in the county.

The lowest level (for the topology geometry layer containing the smallest kinds of features) in a hierarchy is level 0, and successive higher levels are numbered 1, 2, and so on. Topology geometry layers at adjacent levels of a hierarchy have a parent-child relationship. Each topology geometry layer at the higher level is the parent layer for one layer at the lower level, which is its **child layer**. A parent layer can have only one child layer, but a child layer can have one or more parent layers. Using the preceding example, the Counties layer can have only one child layer, Tracts; however, the Tracts layer could have parent layers named Counties and Water Districts (as long as each tract is in only one water district).

Note: Topology geometry layer hierarchy is somewhat similar to network hierarchy, which is described in Section 6.5; however, there are significant differences, and you should not confuse the two. For example, the lowest topology geometry layer hierarchy level is 0, and the lowest network hierarchy level is 1; and in a topology geometry layer hierarchy each parent must have one child and each child can have many parents, while in a network hierarchy each parent can have many children and each child must have one parent.

Figure 1–4 shows the preceding example topology geometry layer hierarchy. Each level of the hierarchy shows the level number and the topology geometry layer in that level.

Figure 1–4 Topology Geometry Layer Hierarchy



To model topology geometry layers as hierarchical, specify the child layer in the `child_layer_id` parameter when you call the SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER procedure to add a parent topology geometry layer to the topology. Add the lowest-level (level 0) topology geometry layer first; then add the level 1 layer, specifying the level 0 layer as its child; then add the level 2 layer, specifying the level 1 layer as its child; and so on. Example 1–1 shows five topology geometry layers being added so that the 5-level hierarchy is established.

Example 1–1 Modeling a Topology Geometry Layer Hierarchy

```
-- Create the topology. (Null SRID in this example.)
EXECUTE SDO_TOPO.CREATE_TOPOLOGY('LAND_USE_HIER', 0.00005);

-- Create feature tables.
CREATE TABLE land_parcels ( -- Land parcels (selected faces)
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

CREATE TABLE block_groups (
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

CREATE TABLE tracts (
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

CREATE TABLE counties (
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

CREATE TABLE states (
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

-- (Other steps not shown here, such as populating the feature tables
-- and initializing the metadata.)
   .
   .
   .
-- Associate feature tables with the topology; include hierarchy information.

DECLARE
  land_parcels_id NUMBER;
  block_groups_id NUMBER;
  tracts_id NUMBER;
  counties_id NUMBER;
BEGIN
SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'LAND_PARCELS',
  'FEATURE','POLYGON');
SELECT tg_layer_id INTO land_parcels_id FROM user_sdo_topo_info
  WHERE topology = 'LAND_USE_HIER' AND table_name = 'LAND_PARCELS';
SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'BLOCK_GROUPS',
  'FEATURE','POLYGON', NULL, land_parcels_id);
SELECT tg_layer_id INTO block_groups_id FROM user_sdo_topo_info
```

```
  WHERE topology = 'LAND_USE_HIER' AND table_name = 'BLOCK_GROUPS';
SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'TRACTS',
  'FEATURE','POLYGON', NULL, block_groups_id);
SELECT tg_layer_id INTO tracts_id FROM user_sdo_topo_info
  WHERE topology = 'LAND_USE_HIER' AND table_name = 'TRACTS';
SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'COUNTIES',
  'FEATURE','POLYGON', NULL, tracts_id);
SELECT tg_layer_id INTO counties_id FROM user_sdo_topo_info
  WHERE topology = 'LAND_USE_HIER' AND table_name = 'COUNTIES';
SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'STATES',
  'FEATURE','POLYGON', NULL, counties_id);
END;
/
```

To insert or update topology geometry objects in feature tables for parent levels in a hierarchy, use the forms of the SDO_TOPO_GEOMETRY constructor that include attributes of type SDO_TGL_OBJECT_ARRAY (as opposed to SDO_TOPO_OBJECT_ARRAY). Feature tables are described in Section 1.3, and SDO_TOPO_GEOMETRY constructors are described in Section 1.6.2.

Note: The TOPO_ID and TOPO_TYPE attributes in the relationship information table have special meanings when applied to parent layers in a topology with a topology geometry layer hierarchy. See the explanations of these attributes in Table 1–5 in Section 1.5.4.

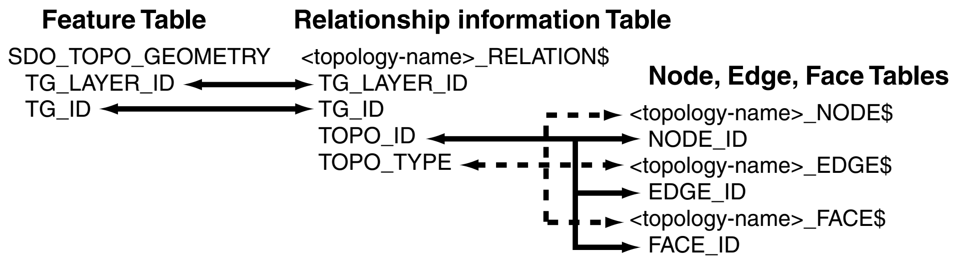
1.5 Topology Data Model Tables

To use the Spatial topology capabilities, you must first insert data into special edge, node, and face tables, which are created by Spatial when you create a topology. The edge, node, and face tables are described in Section 1.5.1, Section 1.5.2, and Section 1.5.3, respectively.

Spatial automatically maintains a relationship information (<topology-name>_RELATION$) table for each topology, which is created the first time that a feature table is associated with a topology (that is, at the first call to the SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER procedure that specifies the topology). The relationship information table is described in Section 1.5.4.

Figure 1–5 shows the role of the relationship information table in connecting information in a feature table with information in its associated node, edge, or face table.

Figure 1–5 Mapping Between Feature Tables and Topology Tables



As shown in Figure 1–5, the mapping between feature tables and the topology node, edge, and face tables occurs through the <topology-name>_RELATION$ table. In particular:

- Each feature table includes a column of type SDO_TOPO_GEOMETRY. This type includes a TG_LAYER_ID attribute (the unique ID assigned by Oracle Spatial Topology Manager when the layer is created), as well as a TG_ID attribute (the unique ID assigned to each feature in a layer). The values in these two columns have corresponding values in the TG_LAYER_ID and TG_ID columns in the <topology-name>_RELATION$ table.
- Each feature has one or more rows in the <topology-name>_RELATION$ table.
- Given the TG_LAYER_ID and TG_ID values for a feature, the set of nodes, faces, and edges associated with the feature can be determined by matching the TOPO_ID value (the node, edge, or face ID) in the <topology-name>_RELATION$ table with the corresponding ID value in the <topology-name>_NODE$, <topology-name>_EDGE$, or <topology-name>_FACE$ table.

1.5.1 Edge Information Table

You must store information about the edges in a topology in the <topology-name>_EDGE$ table, where <*topology-name*> is the name of the topology as specified in the call to the SDO_TOPO.CREATE_TOPOLOGY procedure. Each edge information table has the columns shown in Table 1–1.

Table 1–1 Columns in the <topology-name>_EDGE$ Table

Column Name	Data Type	Description
EDGE_ID	NUMBER	Unique ID number for this edge.
START_NODE_ID	NUMBER	ID number of the start node for this edge.

Table 1–1 (Cont.) Columns in the <topology-name>_EDGE$ Table

Column Name	Data Type	Description
END_NODE_ID	NUMBER	ID number of the end node for this edge.
NEXT_LEFT_EDGE_ID	NUMBER	ID number (signed) of the next left edge for this edge.
PREV_LEFT_EDGE_ID	NUMBER	ID number (signed) of the previous left edge for this edge.
NEXT_RIGHT_EDGE_ID	NUMBER	ID number (signed) of the next right edge for this edge.
PREV_RIGHT_EDGE_ID	NUMBER	ID number (signed) of the previous right edge for this edge.
LEFT_FACE_ID	NUMBER	ID number of the left face for this edge.
RIGHT_FACE_ID	NUMBER	ID number of the right face for this edge.
GEOMETRY	SDO_GEOMETRY	Geometry object (line string) representing this edge.

Figure 1–6 shows nodes, edges, and faces that illustrate the relationships among the various ID columns in the edge information table. (In Figure 1–6, thick lines show the edges, and thin lines with arrowheads show the direction of each edge.)

Figure 1–6 Nodes, Edges, and Faces

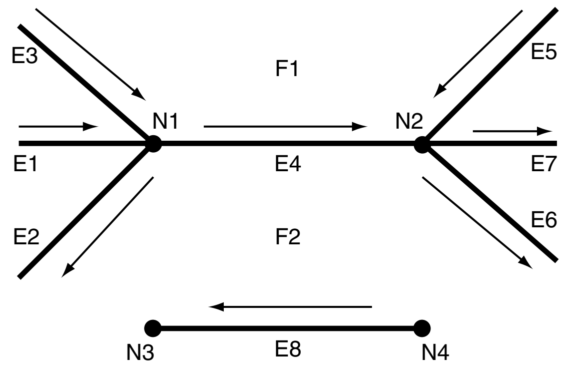


Table 1–2 shows the ID column values in the edge information table for edges E4 and E8 in Figure 1–6. (For clarity, Table 1–2 shows ID column values with alphabetical characters, such as E4 and N1; however, the ID columns actually

contain numeric values only, specifically the numeric ID value associated with each named object.)

Table 1–2 Edge Table ID Column Values

EDGE_ ID	START_ NODE_ ID	END_ NODE_ ID	NEXT_ LEFT_ EDGE_ ID	PREV_ LEFT_ EDGE_ ID	NEXT_ RIGHT_ EDGE_ ID	PREV_ RIGHT_ EDGE_ ID	LEFT_ FACE_ ID	RIGHT_ FACE_ ID
E4	N1	N2	-E5	E3	E2	-E6	F1	F2
E8	N4	N3	-E8	-E8	E8	E8	F2	F2

In Figure 1–6 and Table 1–2:

- The start node and end node for edge E4 are N1 and N2, respectively. The next left edge for edge E4 is E5, but its direction is the opposite of edge E4, and therefore the next left edge for E4 is stored as -E5 (negative E5).
- The previous left edge for edge E4 is E3, and because it has the same direction as edge E4, the previous left edge for E4 is stored as E3.
- The next right face is determined using the negative directed edge of E4. This can be viewed as reversing the edge direction and taking the next left edge and previous left edge. In this case, the next right edge is E2 and the previous right edge is -E6 (the direction of edge E6 is opposite the negative direction of edge E4). For edge E4, the left face is F1 and the right face is F2.
- Edges E1 and E7 are neither leftmost nor rightmost edges with respect to edge E4, and therefore they do not appear in the edge table row associated with edge E4.

1.5.2 Node Information Table

You must store information about the nodes in a topology in the <topology-name>_NODE$ table, where <*topology-name*> is the name of the topology as specified in the call to the SDO_TOPO.CREATE_TOPOLOGY procedure. Each node information table has the columns shown in Table 1–3.

Table 1–3 Columns in the <topology-name>_NODE$ Table

Column Name	Data Type	Description
NODE_ID	NUMBER	Unique ID number for this node.
EDGE_ID	NUMBER	ID number (signed) of the edge (if any) associated with this node.
FACE_ID	NUMBER	ID number of the face (if any) associated with this node.
GEOMETRY	SDO_GEOMETRY	Geometry object (point) representing this node.

1.5.3 Face Information Table

You must store information about the faces in a topology in the <topology-name>_FACE$ table, where <*topology-name*> is the name of the topology as specified in the call to the SDO_TOPO.CREATE_TOPOLOGY procedure. Each face information table has the columns shown in Table 1–4.

Table 1–4 Columns in the <topology-name>_FACE$ Table

Column Name	Data Type	Description
FACE_ID	NUMBER	Unique ID number for this face.
BOUNDARY_EDGE_ID	NUMBER	ID number of the boundary edge for this face. The sign of this number (which is ignored for use as a key) indicates which orientation is being used for this boundary component (positive numbers indicate the left of the edge, and negative numbers indicate the right of the edge).
ISLAND_EDGE_ID_LIST	SDO_LIST_TYPE	Island edges (if any) in this face. (The SDO_LIST_TYPE type is described in Section 1.6.4.)
ISLAND_NODE_ID_LIST	SDO_LIST_TYPE	Island nodes (if any) in this face. (The SDO_LIST_TYPE type is described in Section 1.6.4.)
MBR_GEOMETRY	SDO_GEOMETRY	Minimum bounding rectangle (MBR) that encloses this face. (This is not required. However, if the MBR is specified and if a spatial R-tree index is defined on this geometry, the face can be retrieved more efficiently.)

1.5.4 Relationship Information Table

As you work with topology objects, Spatial automatically maintains information about each object in <topology-name>_RELATION$ tables, where <*topology-name*> is the name of the topology and there is one such table for each topology. Each row in the table uniquely identifies a topology geometry with respect to its topology geometry layer and topology. Each relationship information table has the columns shown in Table 1–5.

Table 1–5 Columns in the <topology-name>_RELATION$ Table

Column Name	Data Type	Description
TG_LAYER_ID	NUMBER	ID number of the topology geometry layer to which the topology geometry belongs.
TG_ID	NUMBER	ID number of the topology geometry.
TOPO_ID	NUMBER	For a topology that does not have a topology geometry layer hierarchy or for the lowest level (level 0) in the hierarchy: ID number of a topological element in the topology geometry. For a level higher than 0 in the hierarchy: level number in the hierarchy of the topology geometry layer.
TOPO_TYPE	NUMBER	For a topology that does not have a topology geometry layer hierarchy or for the lowest level (level 0) in the hierarchy: type of topology: 1 = node, 2 = edge, 3 = face. For a level higher than 0 in the hierarchy: ID number of a topological element in the topology geometry.
TOPO_ATTRIBUTE	VARCHAR2	Reserved for Oracle use.

1.5.5 History Information Table

When a topology editing operation causes an insert or delete operation on an edge or face information table, Spatial automatically maintains information about these operations in <topology-name>_HISTORY$ tables, where <*topology-name*> is the name of the topology and there is one such table for each topology. Each row in the table uniquely identifies an editing operation on a topology object. (Topology editing is discussed in Chapter 2.) Each history information table has the columns shown in Table 1–6.

Table 1–6 Columns in the <topology-name>_HISTORY$ Table

Column Name	Data Type	Description
TOPO_TX_ID	NUMBER	ID number of the transaction that was started by a call to the SDO_TOPO_MAP.LOAD_TOPO_MAP function or to the loadWindow or loadTopology Java method. Each transaction can consist of several editing operations.
TOPO_SEQUENCE	NUMBER	Sequence number assigned to an editing operation within the transaction.
TOPOLOGY	VARCHAR2	Name of the topology containing the objects being edited.
TOPO_ID	NUMBER	ID number of a topological element in the topology geometry.
TOPO_TYPE	NUMBER	Type of topology: 1 = node, 2 = edge, 3 = face.
TOPO_OP	VARCHAR2	Type of editing operation that was performed on the topology object: `I` for insert or `D` for delete.
PARENT_ID	NUMBER	For an insert operation, the ID of the parent topological element from which the current topological element is derived; for a delete operation, the ID of the resulting topological element.

Consider the following examples:

- Adding a node to break edge E2, generating edge E3: The TOPO_ID value of the new edge is the ID of E3, the TOPO_TYPE value is 2, the PARENT_ID value is the ID of E2, and the TOPO_OP value is `I`.
- Deleting a node to merge edges E6 and E7, resulting in E7: The TOPO_ID value is the ID of E6, the TOPO_TYPE value is 2, the PARENT_ID value is the ID of E7, and the TOPO_OP value is `D`.

1.6 Topology Data Types

The main data type associated with the topology data model is SDO_TOPO_GEOMETRY, which describes a topology geometry. The SDO_TOPO_GEOMETRY type has several constructors and one member function. This section describes the topology model types, constructors, and member functions.

1.6.1 SDO_TOPO_GEOMETRY Type

The description of a topology geometry is stored in a single row, in a single column of object type SDO_TOPO_GEOMETRY in a user-defined table. The object type SDO_TOPO_GEOMETRY is defined as:

```
CREATE TYPE sdo_topo_geometry AS OBJECT
  (tg_type      NUMBER,
   tg_id        NUMBER,
   tg_layer_id  NUMBER,
   topology_id  NUMBER);
```

The SDO_TOPO_GEOMETRY type has the attributes shown in Table 1–7.

Table 1–7 SDO_TOPO_GEOMETRY Type Attributes

Attribute	Explanation
TG_TYPE	Type of topology geometry: 1 = point, 2 = line string, 3 = polygon or multipolygon, 4 = heterogeneous collection. Note: Most real world topology geometries are one of the *multi* types.
TG_ID	Unique ID number (generated by Spatial) for the topology geometry.
TG_LAYER_ID	ID number for the topology geometry layer to which the topology geometry belongs. (This number is generated by Spatial, and it is unique within the topology geometry layer.)
TOPOLOGY_ID	Unique ID number (generated by Spatial) for the topology.

Each topology geometry in a topology is uniquely identified by the combination of its TG_ID and TG_LAYER_ID values.

You can use an attribute name in a query on an object of SDO_TOPO_GEOMETRY. Example 1–2 shows SELECT statements that query each attribute of the FEATURE column of the CITY_STREETS table, which is defined in Example 1–8 in Section 1.11.

Example 1–2 SDO_TOPO_GEOMETRY Attributes in Queries

```
SELECT s.feature.tg_type FROM city_streets s;
SELECT s.feature.tg_id FROM city_streets s;
SELECT s.feature.tg_layer_id FROM city_streets s;
SELECT s.feature.topology_id FROM city_streets s;
```

1.6.2 SDO_TOPO_GEOMETRY Constructors

The SDO_TOPO_GEOMETRY type has constructors for inserting and updating topology geometry objects. The constructor format to use for either type of operation (insert or update) depends on whether or not the operation affects a parent level in a topology geometry layer hierarchy:

- To insert and update topology geometry objects when the topology does not have a topology geometry layer hierarchy or when the operation affects the lowest level (level 0) in the hierarchy, use constructors that specify the lowest-level topology objects (nodes, edges, and faces). These constructors have at least one attribute of type SDO_TOPO_OBJECT_ARRAY and no attributes of type SDO_TGL_OBJECT_ARRAY. (Topology geometry layer hierarchy is explained in Section 1.4.)
- To insert and update topology geometry layers when the topology has a topology geometry layer hierarchy and the operation affects a level other than the lowest in the hierarchy, use constructors that specify elements in the child level. These constructors have at least one attribute of type SDO_TGL_OBJECT_ARRAY and no attributes of type SDO_TOPO_OBJECT_ARRAY.

For specifying either lowest-level objects or child-level objects, there are two constructors for insert operations and two constructors for update operations. For each type of operation (insert or update), one constructor format specifies the topology geometry layer by its ID value and the other format specifies the layer by the combination of table name and column name.

This section describes the available SDO_TOPO_GEOMETRY constructors.

1.6.2.1 Constructors for Insert Operations into the Lowest Level

The SDO_TOPO_GEOMETRY type has the following constructors that you can use in INSERT statements to create new topology geometry objects when the topology does not have a topology geometry layer hierarchy or when the operation affects the lowest level (level 0) in the hierarchy:

```
SDO_TOPO_GEOMETRY (topology     VARCHAR2,
                   tg_type      NUMBER,
                   tg_layer_id  NUMBER,
                   topo_ids     SDO_TOPO_OBJECT_ARRAY)

SDO_TOPO_GEOMETRY (topology      VARCHAR2,
                   table_name    VARCHAR2,
                   column_name   VARCHAR2,
                   tg_type       NUMBER,
                    topo_ids      SDO_TOPO_OBJECT_ARRAY)
```

The SDO_TOPO_OBJECT_ARRAY type is defined as a VARRAY of SDO_TOPO_OBJECT objects.

The SDO_TOPO_OBJECT type has the following two attributes:

```
(topo_id NUMBER, topo_type NUMBER)
```

The TG_TYPE and TOPO_IDS attribute values must be within the range of values from the <topology-name>_RELATION$ table (described in Section 1.5.4) for the specified topology.

Example 1–3 shows two SDO_TOPO_GEOMETRY constructors, one in each format. Each constructor inserts a topology geometry into the LAND_PARCELS table, which is defined in Example 1–8 in Section 1.11.

Example 1–3 INSERT Using Constructor with SDO_TOPO_OBJECT_ARRAY

```
INSERT INTO land_parcels VALUES ('P1', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 3), -- face_id = 3
      SDO_TOPO_OBJECT (6, 3))) -- face_id = 6
);

INSERT INTO land_parcels VALUES ('P1A', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    'LAND_PARCELS', -- Table name
    'FEATURE', -- Column name
    3, -- Topology geometry type (polygon/multipolygon)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 3), -- face_id = 3
      SDO_TOPO_OBJECT (6, 3))) -- face_id = 6
);
```

1.6.2.2 Constructors for Insert Operations into a Parent Level

The SDO_TOPO_GEOMETRY type has the following constructors that you can use in INSERT statements into a feature table associated with a parent level in a topology that has a topology geometry layer hierarchy:

```
SDO_TOPO_GEOMETRY (topology     VARCHAR2,
                   tg_type      NUMBER,
                   tg_layer_id  NUMBER,
                   topo_ids     SDO_TGL_OBJECT_ARRAY)

SDO_TOPO_GEOMETRY (topology      VARCHAR2,
                   table_name    VARCHAR2,
                   column_name   VARCHAR2,
                   tg_type       NUMBER,
                   topo_ids      SDO_TGL_OBJECT_ARRAY)
```

The SDO_TGL_OBJECT_ARRAY type is defined as a VARRAY of SDO_TGL_
OBJECT objects.

The SDO_TGL_OBJECT type has the following two attributes:

```
(tgl_id NUMBER, tg_id NUMBER)
```

Example 1–4 shows an SDO_TOPO_GEOMETRY constructor that inserts a row into the BLOCK_GROUPS table, which is the feature table for the Block Groups level in the topology geometry layer hierarchy. The Block Groups level is the parent of the Land Parcels level at the bottom of the hierarchy.

Example 1–4   INSERT Using Constructor with SDO_TGL_OBJECT_ARRAY

```
INSERT INTO block_groups VALUES ('BG1', -- Feature name
  SDO_TOPO_GEOMETRY('LAND_USE_HIER',
    3, -- Topology geometry type (polygon/multipolygon)
    2, -- TG_LAYER_ID for block groups (from ALL_SDO_TOPO_METADATA)
    SDO_TGL_OBJECT_ARRAY (
      SDO_TGL_OBJECT (1, 1), -- land parcel ID = 1
      SDO_TGL_OBJECT (12, 2))) -- land parcel ID = 2
);
```

1.6.2.3 Constructors for Update Operations into the Lowest Level

The SDO_TOPO_GEOMETRY type has the following constructors that you can use in UPDATE statements to modify existing topology geometry objects when the topology does not have a topology geometry layer hierarchy or when the operation affects the lowest level (level 0) in the hierarchy:

```
SDO_TOPO_GEOMETRY (topology         VARCHAR2,
                   tg_type          NUMBER,
                   tg_layer_id      NUMBER,
                   add_topo_ids     SDO_TOPO_OBJECT_ARRAY,
                   delete_topo_ids  SDO_TOPO_OBJECT_ARRAY)
```

```
SDO_TOPO_GEOMETRY (topology         VARCHAR2,
                   table_name       VARCHAR2,
                   column_name      VARCHAR2,
                   tg_type          NUMBER,
                   add_topo_ids     SDO_TOPO_OBJECT_ARRAY,
                   delete_topo_ids  SDO_TOPO_OBJECT_ARRAY)
```

For example, you could use one of these constructor formats to add an edge to a linear feature or to remove an obsolete edge from a feature.

The SDO_TOPO_OBJECT_ARRAY type definition and the requirements for the TG_TYPE and TOPO_IDS attribute values are as described in Section 1.6.2.1.

You can specify values for both the ADD_TOPO_IDS and DELETE_TOPO_IDS attributes, or you can specify values for one attribute and specify the other as null; however, you cannot specify null values for both ADD_TOPO_IDS and DELETE_TOPO_IDS.

Example 1–5 shows two SDO_TOPO_GEOMETRY constructors, one in each format. Each constructor removes two faces from the `CITY_DATA` topology in the LAND_PARCELS table, which is defined in Example 1–8 in Section 1.11.

Example 1–5 UPDATE Using Constructor with SDO_TOPO_OBJECT_ARRAY

```
UPDATE land_parcels l SET l.feature = SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    NULL, -- No topology objects to be added
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 3), -- face_id = 3
      SDO_TOPO_OBJECT (6, 3))) -- face_id = 6
WHERE l.feature_name = 'P1';

UPDATE land_parcels l SET l.feature = SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    'LAND_PARCELS', -- Table name
    'FEATURE', -- Column name
    3, -- Topology geometry type (polygon/multipolygon)
    NULL, -- No topology objects to be added
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 3), -- face_id = 3
      SDO_TOPO_OBJECT (6, 3))) -- face_id = 6
WHERE l.feature_name = 'P1A';
```

1.6.2.4 Constructors for Update Operations into a Parent Level

The SDO_TOPO_GEOMETRY type has the following constructors that you can use in UPDATE statements affecting a feature table associated with a parent level in a topology that has a topology geometry layer hierarchy:

```
SDO_TOPO_GEOMETRY (topology         VARCHAR2,
                   tg_type          NUMBER,
                   tg_layer_id      NUMBER,
                   add_topo_ids     SDO_TGL_OBJECT_ARRAY,
                   delete_topo_ids  SDO_TGL_OBJECT_ARRAY)

SDO_TOPO_GEOMETRY (topology         VARCHAR2,
                   table_name       VARCHAR2,
                   column_name      VARCHAR2,
                   tg_type          NUMBER,
                   add_topo_ids     SDO_TGL_OBJECT_ARRAY,
                   delete_topo_ids  SDO_TGL_OBJECT_ARRAY)
```

For example, you could use one of these constructor formats to add an edge to a linear feature or to remove an obsolete edge from a feature.

The SDO_TGL_OBJECT_ARRAY type definition and the requirements for its attribute values are as described in Section 1.6.2.2.

You can specify values for both the ADD_TOPO_IDS and DELETE_TOPO_IDS attributes, or you can specify values for one attribute and specify the other as null; however, you cannot specify null values for both ADD_TOPO_IDS and DELETE_TOPO_IDS.

Example 1–6 shows two SDO_TOPO_GEOMETRY constructors, one in each format. Each constructor deletes the land parcel with the ID value of 2 from two features (named `BG1` and `BG1A` and that have the same definition) from the `CITY_DATA` topology in the BLOCK_GROUPS table, which is the feature table for the Block Groups level in the topology geometry layer hierarchy. The Block Groups level is the parent of the Land Parcels level at the bottom of the hierarchy.

Example 1–6 UPDATE Using Constructor with SDO_TGL_OBJECT_ARRAY

```
UPDATE block_groups b SET b.feature = SDO_TOPO_GEOMETRY(
  'LAND_USE_HIER',
  3, -- Topology geometry type (polygon/multipolygon)
  2, -- TG_LAYER_ID for block groups (from ALL_SDO_TOPO_METADATA)
  null, -- No IDs to add
  SDO_TGL_OBJECT_ARRAY (
    SDO_TGL_OBJECT (1, 2)) -- land parcel ID = 2
  )
WHERE b.feature_name = 'BG1';

UPDATE block_groups b SET b.feature = SDO_TOPO_GEOMETRY(
  'LAND_USE_HIER',
  'BLOCK_GROUPS', -- Feature table
  'FEATURE', -- Feature column
  3, -- Topology geometry type (polygon/multipolygon)
  null, -- No IDs to add
  SDO_TGL_OBJECT_ARRAY (
    SDO_TGL_OBJECT (1, 2)) -- land parcel ID = 2
  )
WHERE b.feature_name = 'BG1A';
```

1.6.3 GET_GEOMETRY Member Function

The SDO_TOPO_GEOMETRY type has a member function GET_GEOMETRY, which you can use to return the SDO_GEOMETRY object for the topology geometry object.

Example 1–7 uses the GET_GEOMETRY member function to return the SDO_GEOMETRY object for the topology geometry object associated with the land parcel named `P1`.

Example 1–7 GET_GEOMETRY Member Function

```
SELECT l.feature_name, l.feature.get_geometry()
  FROM land_parcels l WHERE l.feature_name = 'P1';

FEATURE_NAME
------------------------------
L.FEATURE.GET_GEOMETRY()(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
--------------------------------------------------------------------------------
P1
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 3, 1), SDO_ORDINATE_ARRAY(
21, 14, 21, 22, 9, 22, 9, 14, 9, 6, 21, 6, 21, 14))
```

1.6.4 SDO_LIST_TYPE Type

The SDO_LIST_TYPE type is used to store the EDGE_ID values of island edges and NODE_ID values of island nodes in a face. The SDO_LIST_TYPE type is defined as:

```
CREATE TYPE sdo_list_type as VARRAY(2147483647) OF NUMBER;
```

1.6.5 SDO_EDGE_ARRAY and SDO_NUMBER_ARRAY Types

The SDO_EDGE_ARRAY type is used to specify the coordinates of attached edges affected by a node move operation. The SDO_EDGE_ARRAY type is defined as:

```
CREATE TYPE sdo_edge_array as VARRAY(1000000) OF MDSYS.SDO_NUMBER_ARRAY;
```

The SDO_NUMBER_ARRAY type is a general-purpose type used by Spatial for arrays. The SDO_NUMBER_ARRAY type is defined as:

```
CREATE TYPE sdo_number_array as VARRAY(1048576) OF NUMBER;
```

1.7 Topology Metadata Views

There are two sets of topology metadata views for each schema (user): xxx_SDO_TOPO_INFO and xxx_SDO_TOPO_METADATA, where *xxx* can be USER or ALL. These views are read-only to users; they are created and maintained by Spatial.

The xxx_SDO_TOPO_METADATA views contain the most detailed information, and each xxx_SDO_TOPO_INFO view contains a subset of the information in its corresponding xxx_SDO_TOPO_METADATA view.

1.7.1 xxx_SDO_TOPO_INFO Views

The following views contain basic information about topologies:

- USER_SDO_TOPO_INFO contains topology information for all tables owned by the user.
- ALL_SDO_TOPO_INFO contains topology information for all tables on which the user has SELECT permission.

The USER_SDO_TOPO_INFO and ALL_SDO_TOPO_INFO views contain the same columns, as shown Table 1–8. (The columns are listed in their order in the view definition.)

Table 1–8 Columns in the xxx_SDO_TOPO_INFO Views

Column Name	Data Type	Purpose
OWNER	VARCHAR2	Owner of the topology.
TOPOLOGY	VARCHAR2	Name of the topology.
TOPOLOGY_TYPE	VARCHAR2	Contains PLANAR if the topology can have nodes, edges, and faces. (No other values are supported for the current release.)

Table 1–8 (Cont.) Columns in the xxx_SDO_TOPO_INFO Views

Column Name	Data Type	Purpose
TOLERANCE	NUMBER	Tolerance value associated with topology geometries in the topology. (Tolerance is explained in Chapter 1 of the *Oracle Spatial User's Guide and Reference*.) Oracle Spatial uses the tolerance value in building R-tree indexes on the node, edge, and face tables; the value is also used for any spatial queries that use these tables.
SRID	NUMBER	Coordinate system (spatial reference system) associated with all topology geometry layers in the topology. Is null if no coordinate system is associated; otherwise, it contains a value from the SRID column of the MDSYS.CS_SRS table (described in *Oracle Spatial User's Guide and Reference*).
TABLE_SCHEMA	VARCHAR2	Name of the schema that owns the table containing the topology geometry layer column.
TABLE_NAME	VARCHAR2	Name of the table containing the topology geometry layer column.
COLUMN_NAME	VARCHAR2	Name of the column containing the topology geometry layer data.
TG_LAYER_ID	NUMBER	ID number of the topology geometry layer.
TG_LAYER_TYPE	VARCHAR2	Contains one of the following: POINT, LINE, CURVE, POLYGON, or COLLECTION.
TG_LAYER_LEVEL	NUMBER	Hierarchy level number of this topology geometry layer. (Topology geometry layer hierarchy is explained in Section 1.4.)
CHILD_LAYER_ID	NUMBER	ID number of the topology geometry layer that is the child layer of this layer in the topology geometry layer hierarchy. Null if this layer has no child layer or if the topology does not have a topology geometry layer hierarchy. (Topology geometry layer hierarchy is explained in Section 1.4.)

1.7.2 xxx_SDO_TOPO_METADATA Views

The following views contain detailed information about topologies:

- USER_SDO_TOPO_METADATA contains topology information for all tables owned by the user.

- ALL_SDO_TOPO_METADATA contains topology information for all tables on which the user has SELECT permission.

The USER_SDO_TOPO_METADATA and ALL_SDO_TOPO_METADATA views contain the same columns, as shown Table 1–9. (The columns are listed in their order in the view definition.)

Table 1–9 Columns in the xxx_SDO_TOPO_METADATA Views

Column Name	Data Type	Purpose
OWNER	VARCHAR2	Owner of the topology.
TOPOLOGY	VARCHAR2	Name of the topology.
TOPOLOGY_TYPE	VARCHAR2	Contains PLANAR if the topology can have nodes, edges, and faces. (No other values are supported for the current release.)
TOLERANCE	NUMBER	Tolerance value associated with topology geometries in the topology. (Tolerance is explained in Chapter 1 of *Oracle Spatial User's Guide and Reference*.) Oracle Spatial uses the tolerance value in building R-tree indexes on the node, edge, and face tables; the value is also used for any spatial queries that use these tables.
SRID	NUMBER	Coordinate system (spatial reference system) associated with all topology geometry layers in the topology. Is null if no coordinate system is associated; otherwise, contains a value from the SRID column of the MDSYS.CS_SRS table (described in *Oracle Spatial User's Guide and Reference*).
TABLE_SCHEMA	VARCHAR2	Name of the schema that owns the table containing the topology geometry layer column.
TABLE_NAME	VARCHAR2	Name of the table containing the topology geometry layer column.
COLUMN_NAME	VARCHAR2	Name of the column containing the topology geometry layer data.
TG_LAYER_ID	NUMBER	ID number of the topology geometry layer.
TG_LAYER_TYPE	VARCHAR2	Contains one of the following: POINT, LINE, CURVE, or POLYGON.
TG_LAYER_LEVEL	NUMBER	Hierarchy level number of this topology geometry layer. (Topology geometry layer hierarchy is explained in Section 1.4.)

Table 1–9 (Cont.) Columns in the xxx_SDO_TOPO_METADATA Views

Column Name	Data Type	Purpose
CHILD_LAYER_ID	NUMBER	ID number of the topology geometry layer that is the child layer of this layer in the topology geometry layer hierarchy. Null if this layer has no child layer or if the topology does not have a geometry layer hierarchy. (Topology geometry layer hierarchy is explained in Section 1.4.)
NODE_SEQUENCE	VARCHAR2	Name of the sequence containing the next available node ID number.
EDGE_SEQUENCE	VARCHAR2	Name of the sequence containing the next available edge ID number.
FACE_SEQUENCE	VARCHAR2	Name of the sequence containing the next available face ID number.
TG_SEQUENCE	VARCHAR2	Name of the sequence containing the next available topology geometry ID number.

1.8 Topology Application Programming Interface

The topology data model application programming interface (API) consists of the following:

- Subprograms (PL/SQL functions and procedures) in the SDO_TOPO package (described in Chapter 3) and the SDO_TOPO_MAP package (described in Chapter 4)
- SDO_ANYINTERACT operator (described in Chapter 5)
- Java API (described in Section 1.8.1)

1.8.1 Topology Data Model Java Interface

The Java client interface for the topology data model consists of the following classes:

- `TopoMap`: class that stores edges, nodes, and faces, and provides methods for adding and deleting elements while maintaining topological consistency both in the cache and in the underlying database tables
- `Edge`: class for an edge
- `Face`: class for a face

- `Node`: class for a node
- `Point2DD`: class for a point
- `IntArrayList`: class for the `int` data type
- `InvalidTopoOperationException`: class for the invalid topology operation exception
- `TopoValidationException`: class for the topology validation failure exception
- `TopoEntityNotFoundException`: class for the entity not found exception

For detailed reference information about the topology data model classes, as well as some usage information about the Java API, see the Javadoc-generated API documentation: open `index.html` in a directory that includes the path `sdotopo/doc/javadoc`.

1.9 Exporting and Importing Topology Data

To export topology data from one database and import it into another database, follow the steps in this section.

In the database with the topology data to be exported, export the topology data by exporting all topology tables, including the feature tables.

In the database into which to import the topology data:

1. Import the tables from the .dmp file that you created when you exported the topology data.
2. Call the SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER procedure to add each topology geometry layer to the topology.
3. Call the SDO_TOPO.INITIALIZE_METADATA procedure to initialize the topology metadata.

1.10 Function-Based Indexes Not Supported

You cannot create a function-based index on a column of type SDO_TOPO_GEOMETRY. (Function-based indexes are explained in *Oracle Database Application Developer's Guide - Fundamentals* and *Oracle Database Administrator's Guide*.)

1.11 Topology Example (PL/SQL)

This section presents a simplified PL/SQL example that performs topology data model operations. It refers to concepts that are explained in this chapter. It uses SDO_TOPO functions and procedures, which are documented in Chapter 3, and the SDO_ANYINTERACT operator, which is documented in Chapter 5.

Example 1–8 uses the topology shown in Figure 1–1 in Section 1.2, and the features shown in Figure 1–3 in Section 1.3.1.

Example 1–8 Topology Example (PL/SQL)

```
------------------------------
-- Main steps for using the topology data model
------------------------------
-- 1. Create a topology.
-- 2. Load (normally bulk-load) topology data (node, edge, and face tables).
-- 3. Create feature tables.
-- 4. Associate feature tables with the topology.
-- 5. Initialize topology metadata.
-- 6. Load feature tables using the SDO_TOPO_GEOMETRY constructor.
-- 7. Query the data.
-- 8. Optionally, edit data using the Java API.

-- 1. Create the topology. (Null SRID in this example.)
EXECUTE SDO_TOPO.CREATE_TOPOLOGY('CITY_DATA', 0.00005);

-- 2. Load topology data (node, edge, and face tables).
--  Use INSERT statements here instead of a bulk-load utility.

-- 2A. Insert data into <topology_name>_EDGE$ table.

-- E1
INSERT INTO city_data_edge$ VALUES(1, 1, 1, 1, 1, -1, -1, 1, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(8,30, 16,30, 16,38, 3,38, 3,30, 8,30)));
-- E2
INSERT INTO city_data_edge$ VALUES(2, 2, 2, 3, -3, -2, -2, 2, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(25,30, 31,30, 31,40, 17,40, 17,30, 25,30)));
-- E3
INSERT INTO city_data_edge$ VALUES(3, 2, 3, -3, 2, 2, 3, 2, 2,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(25,30, 25,35)));
-- E4
```

```
INSERT INTO city_data_edge$ VALUES(4, 5, 6, -5, -4, 4, 5, -1, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(36,38, 38,35, 41,34, 42,33, 45,32, 47,28, 50,28, 52,32,
57,33)));
-- E5
INSERT INTO city_data_edge$ VALUES(5, 7, 6, -4, -5, 5, 4, -1, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(41,40, 45,40, 47,42, 62,41, 61,38, 59,39, 57,36,
57,33)));
-- E6
INSERT INTO city_data_edge$ VALUES(6, 16, 17, 7, 21, -21, 19, -1, 3,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(9,22, 21,22)));
-- E7
INSERT INTO city_data_edge$ VALUES(7, 17, 18, 8, 6, -19, 17, -1, 4,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(21,22, 35,22)));
-- E8
INSERT INTO city_data_edge$ VALUES(8, 18, 19, -15, 7, -17, 15, -1, 5,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(35,22, 47,22)));
-- E9
INSERT INTO city_data_edge$ VALUES(9, 15, 14, 19, -21, -22, 20, 3, 6,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(9,14, 21,14)));
-- E10
INSERT INTO city_data_edge$ VALUES(10, 13, 14, -20, 18, 17, -19, 7, 4,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(35,14, 21,14)));
-- E11
INSERT INTO city_data_edge$ VALUES(11, 13, 12, 15, -17, -18, 16, 5, 8,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(35,14, 47,14)));
-- E12
INSERT INTO city_data_edge$ VALUES(12, 8, 9, 20, -22, 22, -13, 6, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(9,6, 21,6)));
-- E13
INSERT INTO city_data_edge$ VALUES(13, 9, 10, 18, -20, -12, -14, 7, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(21,6, 35,6)));
-- E14
INSERT INTO city_data_edge$ VALUES(14, 10, 11, 16, -18, -13, -16, 8, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(35,6, 47,6)));
```

```
-- E15
INSERT INTO city_data_edge$ VALUES(15, 12, 19, -8, 11, -16, 8, 5, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(47,14, 47,22)));
-- E16
INSERT INTO city_data_edge$ VALUES(16, 11, 12, -11, 14, -14, -15, 8, -1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(47,6, 47,14)));
-- E17
INSERT INTO city_data_edge$ VALUES(17, 13, 18, -7, -10, 11, -8, 4, 5,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(35,14, 35,22)));
-- E18
INSERT INTO city_data_edge$ VALUES(18, 10, 13, 10, 13, 14, -11, 7, 8,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(35,6, 35,14)));
-- E19
INSERT INTO city_data_edge$ VALUES(19, 14, 17, -6, 9, -10, -7, 3, 4,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(21,14, 21,22)));
-- E20
INSERT INTO city_data_edge$ VALUES(20, 9, 14, -9, 12, 13, 10, 6, 7,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(21,6, 21,14)));
-- E21
INSERT INTO city_data_edge$ VALUES(21, 15, 16, 6, 22, 9, -6, -1, 3,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(9,14, 9,22)));
-- E22
INSERT INTO city_data_edge$ VALUES(22, 8, 15, 21, -12, 12, -9, -1, 6,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(9,6, 9,14)));
-- E25
INSERT INTO city_data_edge$ VALUES(25, 21, 22, -25, -25, 25, 25, 1, 1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(9,35, 13,35)));
-- E26
INSERT INTO city_data_edge$ VALUES(26, 20, 20, 26, 26, -26, -26, 9, 1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(4,31, 7,31, 7,34, 4,34, 4,31)));

-- 2B. Insert data into <topology_name>_NODE$ table.

-- N1
INSERT INTO city_data_node$ VALUES(1, 1, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(8,30,NULL), NULL, NULL));
-- N2
INSERT INTO city_data_node$ VALUES(2, 2, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(25,30,NULL), NULL, NULL));
-- N3
INSERT INTO city_data_node$ VALUES(3, -3, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(25,35,NULL), NULL, NULL));
-- N4
INSERT INTO city_data_node$ VALUES(4, NULL, 2,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(20,37,NULL), NULL, NULL));
-- N5
INSERT INTO city_data_node$ VALUES(5, 4, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(36,38,NULL), NULL, NULL));
-- N6
INSERT INTO city_data_node$ VALUES(6, -4, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(57,33,NULL), NULL, NULL));
-- N7
INSERT INTO city_data_node$ VALUES(7, 5, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(41,40,NULL), NULL, NULL));
-- N8
INSERT INTO city_data_node$ VALUES(8, 12, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(9,6,NULL), NULL, NULL));
-- N9
INSERT INTO city_data_node$ VALUES(9, 20, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(21,6,NULL), NULL, NULL));
-- N10
INSERT INTO city_data_node$ VALUES(10, 18, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(35,6,NULL), NULL, NULL));
-- N11
INSERT INTO city_data_node$ VALUES(11, -14, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(47,6,NULL), NULL, NULL));
-- N12
INSERT INTO city_data_node$ VALUES(12, 15, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(47,14,NULL), NULL, NULL));
-- N13
INSERT INTO city_data_node$ VALUES(13, 17, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(35,14,NULL), NULL, NULL));
-- N14
INSERT INTO city_data_node$ VALUES(14, 19, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(21,14,NULL), NULL, NULL));
-- N15
INSERT INTO city_data_node$ VALUES(15, 21, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(9,14,NULL), NULL, NULL));
-- N16
INSERT INTO city_data_node$ VALUES(16, 6, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(9,22,NULL), NULL, NULL));
-- N17
INSERT INTO city_data_node$ VALUES(17, 7, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(21,22,NULL), NULL, NULL));
-- N18
INSERT INTO city_data_node$ VALUES(18, 8, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(35,22,NULL), NULL, NULL));
-- N19
INSERT INTO city_data_node$ VALUES(19, -15, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(47,22,NULL), NULL, NULL));
-- N20
INSERT INTO city_data_node$ VALUES(20, 26, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(4,31,NULL), NULL, NULL));
-- N21
INSERT INTO city_data_node$ VALUES(21, 25, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(9,35,NULL), NULL, NULL));
-- N22
INSERT INTO city_data_node$ VALUES(22, -25, NULL,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(13,35,NULL), NULL, NULL));

-- 2C. Insert data into <topology_name>_FACE$ table.

-- F0 (id = -1, not 0)
INSERT INTO city_data_face$ VALUES(-1, NULL, SDO_LIST_TYPE(-1, -2, 4, 6),
                                                           SDO_LIST_TYPE(),
NULL);
-- F1
INSERT INTO city_data_face$ VALUES(1, 1, SDO_LIST_TYPE(25), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(3,30, 15,38)));
-- F2
INSERT INTO city_data_face$ VALUES(2, 2, SDO_LIST_TYPE(), SDO_LIST_TYPE(4),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(17,30, 31,40)));
-- F3
INSERT INTO city_data_face$ VALUES(3, 19, SDO_LIST_TYPE(), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(9,14, 21,22)));
-- F4
INSERT INTO city_data_face$ VALUES(4, 17, SDO_LIST_TYPE(), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(21,14, 35,22)));
-- F5
INSERT INTO city_data_face$ VALUES(5, 15, SDO_LIST_TYPE(), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(35,14, 47,22)));
-- F6
INSERT INTO city_data_face$ VALUES(6, 20, SDO_LIST_TYPE(), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(9,6, 21,14)));
-- F7
INSERT INTO city_data_face$ VALUES(7, 10, SDO_LIST_TYPE(), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(21,6, 35,14)));
-- F8
INSERT INTO city_data_face$ VALUES(8, 16, SDO_LIST_TYPE(), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(35,6, 47,14)));
-- F9
INSERT INTO city_data_face$ VALUES(9,26,SDO_LIST_TYPE(), SDO_LIST_TYPE(),
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(4,31, 7,34)));

-- 3. Create feature tables.

CREATE TABLE land_parcels ( -- Land parcels (selected faces)
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

CREATE TABLE city_streets ( -- City streets (selected edges)
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

CREATE TABLE traffic_signs ( -- Traffic signs (selected nodes)
  feature_name VARCHAR2(30) PRIMARY KEY,
  feature SDO_TOPO_GEOMETRY);

-- 4. Associate feature tables with the topology.
--    Add the three topology geometry layers to the CITY_DATA topology.
--    Any order is OK.

EXECUTE SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('CITY_DATA', 'LAND_PARCELS','FEATURE',
'POLYGON');
EXECUTE SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('CITY_DATA', 'TRAFFIC_SIGNS','FEATURE',
'POINT');
EXECUTE SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('CITY_DATA', 'CITY_STREETS',
'FEATURE','LINE');

--  As a result, Spatial generates a unique TG_LAYER_ID for each layer in
--  the topology metadata (USER/ALL_SDO_TOPO_METADATA).
-- 5. Initialize topology metadata.
EXECUTE SDO_TOPO.INITIALIZE_METADATA('CITY_DATA');

-- 6. Load feature tables using the SDO_TOPO_GEOMETRY constructor.

-- Each topology feature can consist of one or more objects (face, edge, node)
-- of an appropriate type. For example, a land parcel can consist of one face,
-- or two or more faces, as specified in the SDO_TOPO_OBJECT_ARRAY.

-- There are typically fewer features than there are faces, nodes, and edges.
-- In this example, the only features are these:
-- Area features (land parcels): P1, P2, P3, P4, P5
-- Point features (traffic signs): S1, S2, S3, S4
-- Linear features (roads/streets): R1, R2, R3, R4

-- 6A. Load LAND_PARCELS table.

-- P1
INSERT INTO land_parcels VALUES ('P1', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 3), -- face_id = 3
      SDO_TOPO_OBJECT (6, 3))) -- face_id = 6
);
-- P2
INSERT INTO land_parcels VALUES ('P2', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (4, 3), -- face_id = 4
      SDO_TOPO_OBJECT (7, 3))) -- face_id = 7
);
-- P3
INSERT INTO land_parcels VALUES ('P3', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (5, 3), -- face_id = 5
      SDO_TOPO_OBJECT (8, 3))) -- face_id = 8
);
-- P4
INSERT INTO land_parcels VALUES ('P4', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (2, 3))) -- face_id = 2
);
-- P5 (Includes F1, but not F9.)
INSERT INTO land_parcels VALUES ('P5', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (1, 3))) -- face_id = 1
);

-- 6B. Load TRAFFIC_SIGNS table.

-- S1
INSERT INTO traffic_signs VALUES ('S1', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    1, -- Topology geometry type (point)
    2, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (14, 1))) -- node_id = 14
);
-- S2
INSERT INTO traffic_signs VALUES ('S2', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    1, -- Topology geometry type (point)
    2, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (13, 1))) -- node_id = 13
);
-- S3
INSERT INTO traffic_signs VALUES ('S3', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    1, -- Topology geometry type (point)
    2, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (6, 1))) -- node_id = 6
);
-- S4
INSERT INTO traffic_signs VALUES ('S4', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    1, -- Topology geometry type (point)
    2, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (4, 1))) -- node_id = 4
);

-- 6C. Load CITY_STREETS table.
-- (Note: "R" in feature names is for "Road", because "S" is used for signs.)

-- R1
INSERT INTO city_streets VALUES ('R1', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    2, -- Topology geometry type (line string)
    3, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (9, 2),
      SDO_TOPO_OBJECT (-10, 2),
      SDO_TOPO_OBJECT (11, 2))) -- edge_ids = 9, -10, 11
);
-- R2
INSERT INTO city_streets VALUES ('R2', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    2, -- Topology geometry type (line string)
    3, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (4, 2),
      SDO_TOPO_OBJECT (-5, 2))) -- edge_ids = 4, -5
);
-- R3
INSERT INTO city_streets VALUES ('R3', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    2, -- Topology geometry type (line string)
    3, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (25, 2))) -- edge_id = 25
);
-- R4
INSERT INTO city_streets VALUES ('R4', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    2, -- Topology geometry type (line string)
    3, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 2))) -- edge_id = 3
);

-- 7. Query the data.

SELECT a.feature_name, a.feature.tg_id, a.feature.get_geometry()
FROM land_parcels a;

/* Window is city_streets */
SELECT  a.feature_name, b.feature_name
  FROM city_streets b,
     land_parcels a
  WHERE  b.feature_name like 'R%' AND
     sdo_anyinteract(a.feature, b.feature) = 'TRUE'
  ORDER BY b.feature_name, a.feature_name;

-- Find all streets that have any interaction with land parcel P3.
-- (Should return only R1.)
SELECT c.feature_name FROM city_streets c, land_parcels l
  WHERE l.feature_name = 'P3' AND
   SDO_ANYINTERACT (c.feature, l.feature) = 'TRUE';

-- Find all land parcels that have any interaction with traffic sign S1.
-- (Should return P1 and P2.)
SELECT l.feature_name FROM land_parcels l, traffic_signs t
  WHERE t.feature_name = 'S1' AND
   SDO_ANYINTERACT (l.feature, t.feature) = 'TRUE';

-- Get the geometry for land parcel P1.
SELECT l.feature_name, l.feature.get_geometry()
  FROM land_parcels l WHERE l.feature_name = 'P1';

-- Get the boundary of face with face_id 3.
SELECT SDO_TOPO.GET_FACE_BOUNDARY('CITY_DATA', 3) FROM DUAL;
```

```
-- Get the topology objects for land parcel P2.
-- CITY_DATA layer, land parcels (tg_ layer_id = 1), parcel P2 (tg_id = 2)
SELECT SDO_TOPO.GET_TOPO_OBJECTS('CITY_DATA', 1, 2) FROM DUAL;
```

2 Editing Topologies

This chapter explains how to edit node and edge data in a topology. The operations include adding, moving, and removing nodes and edges, and updating the coordinates of an edge.

This chapter explains two approaches to editing topology data, and it explains why one approach (creating and using a special cache) is better in most cases. It also describes the behavior and implications of some major types of editing operations. It contains the following major sections:

- Section 2.1, "Approaches for Editing Topology Data"
- Section 2.2, "Performing Operations on Nodes"
- Section 2.3, "Performing Operations on Edges"

The explanations in this chapter refer mainly to the PL/SQL application programming interface (API) provided in the MDSYS.SDO_TOPO_MAP packages, which is documented in Chapter 4. However, you can also perform topology editing operations using the client-side Java API, which is introduced in Section 1.8.1 and is explained in the Javadoc-generated documentation.

To edit topology data, always use the PL/SQL or Java API. Do not try to perform editing operations by directly modifying the node, edge, or face information tables.

2.1 Approaches for Editing Topology Data

Whenever you need to edit a topology, you can use PL/SQL or Java API. In both cases, Oracle Spatial uses an in-memory topology cache, specifically, a TopoMap object (described in Section 2.1.1):

- If you use the PL/SQL API, you can either explicitly create and use the cache or allow Spatial to create and use the cache automatically.

- If you use the Java API, you must explicitly create and use the cache.

Allowing Spatial to create and manage the cache automatically is simpler, because it involves fewer steps than creating and using a cache. However, because allowing Spatial to create and manage the cache involves more database activity and disk accesses, it is less efficient when you need to edit more than a few topology objects.

2.1.1 TopoMap Objects

A **TopoMap object** is associated with an in-memory cache that is associated with a topology. If you explicitly create and use a cache for editing a topology, you must create a TopoMap object to be associated with a topology, load all or some of the topology into the cache, edit objects, periodically update the topology to write changes to the database, commit the changes made in the cache, and clear the cache.

Although this approach involves more steps than allowing Spatial to create and use the cache automatically, it is much faster and more efficient for most topology editing sessions, which typically affect hundreds or thousands of topology objects. It is the approach shown in most explanations and illustrations.

A TopoMap object can be updatable or read-only, depending on the value of the `allow_updates` parameter when you call the SDO_TOPO_MAP.LOAD_TOPO_MAP procedure. The following procedures set an updatable TopoMap object to be read-only:

- SDO_TOPO_MAP.COMMIT_TOPO_MAP
- SDO_TOPO_MAP.ROLLBACK_TOPO_MAP
- SDO_TOPO_MAP.CLEAR_TOPO_MAP

There can be no more than one updatable TopoMap object in a user session at any time. There can be multiple read-only TopoMap objects.

2.1.2 Specifying the Editing Approach with the Topology Parameter

For many SDO_TOPO_MAP package functions and procedures that edit topologies, such as SDO_TOPO_MAP.ADD_NODE or SDO_TOPO_MAP.MOVE_EDGE, you indicate the approach you are using for editing by specifying either a topology name or a null value for the first parameter, which is named `topology`:

- If you specify a topology name, Spatial checks to see if an updatable TopoMap object already exists in the user session; and if one does not exist, Spatial creates an internal TopoMap object, uses that cache to perform the editing operation, commits the change (or rolls back changes to the savepoint at the beginning of the process if an exception occurred), and deletes the TopoMap object. (If an updatable TopoMap object already exists, an exception is raised.) For example, the following statement removes the node with node ID value 99 from the MY_TOPO topology:

```
CALL SDO_TOPO_MAP.REMOVE_NODE('MY_TOPO', 99);
```

- If you specify a null value, Spatial checks to see if an updatable TopoMap object already exists in the user session; and if one does exist, Spatial performs the operation in the TopoMap object's cache. (If no updatable TopoMap object exists, an exception is raised.) For example, the following statement removes the node with node ID value 99 from the current updatable TopoMap object:

  ```
  CALL SDO_TOPO_MAP.REMOVE_NODE(null, 99);
  ```

2.1.3 Using GET_xxx Topology Functions

Some SDO_TOPO_MAP package functions that get information about topologies have `topology` and `topo_map` as their first two parameters. Examples of such functions are SDO_TOPO_MAP.GET_EDGE_COORDS and SDO_TOPO_MAP.GET_NODE_STAR. To use these functions, specify a valid value for one parameter and a null value for the other parameter, as follows:

- If you specify a valid `topology` parameter value, Spatial retrieves the information for the specified topology. It creates an internal TopoMap object, uses that cache to perform the operation, and deletes the TopoMap object. For example, the following statement returns the edge coordinates of the edge with an ID value of 1 from the CITY_DATA topology:

  ```
  SELECT SDO_TOPO_MAP.GET_EDGE_COORDS('CITY_DATA', null, 1) FROM DUAL;
  ```

- If you specify a null `topology` parameter value and a valid `topo_map` parameter value, Spatial uses the specified TopoMap object (which can be updatable or read-only) to retrieve the information for the specified topology. For example, the following statement returns the edge coordinates of the edge with an ID value of 1 from the CITY_DATA_TOPOMAP TopoMap object:

  ```
  SELECT SDO_TOPO_MAP.GET_EDGE_COORDS(null, 'CITY_DATA_TOPOMAP', 1) FROM DUAL;
  ```

- If you specify a null or invalid value for both the `topology` and `topo_map` parameters, an exception is raised.

Some SDO_TOPO_MAP package functions that get information about topology editing operations have no parameters. Examples of such functions are SDO_TOPO_MAP.GET_FACE_ADDITIONS and SDO_TOPO_MAP.GET_NODE_

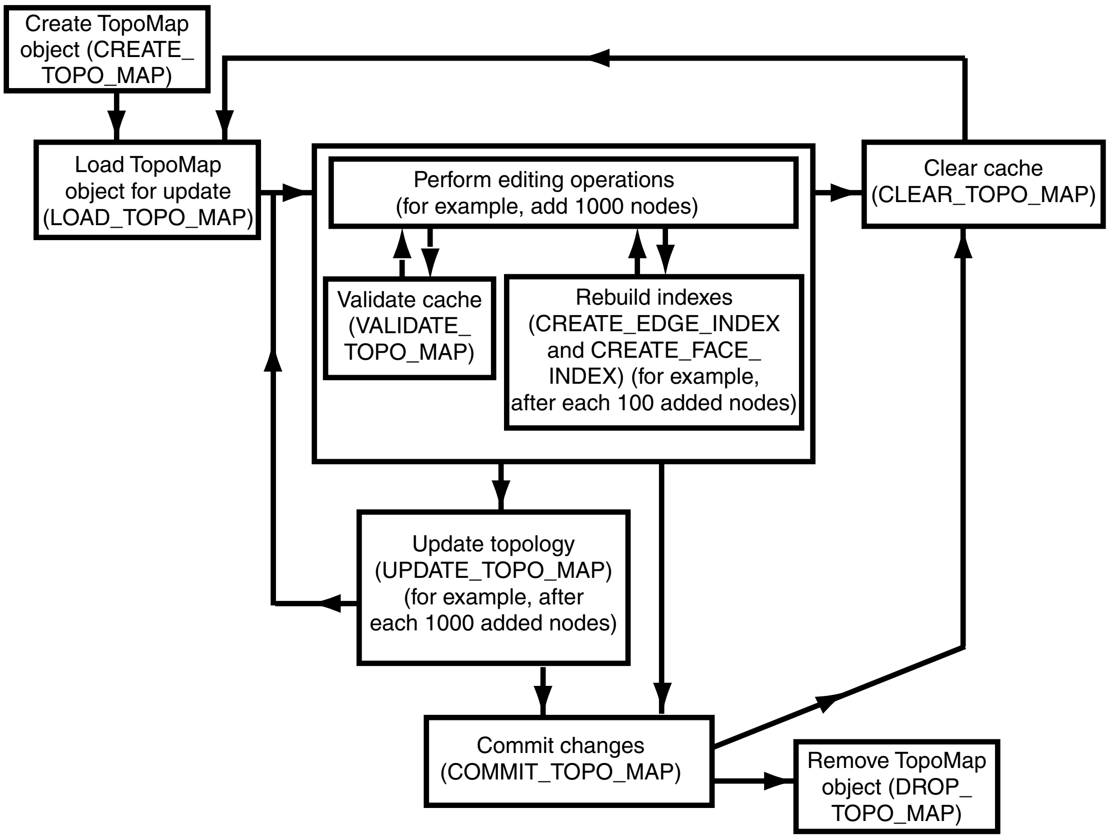
CHANGES. These functions use the current updatable TopoMap object. If no updatable TopoMap object exists, an exception is raised. For example, the following statement returns an SDO_NUMBER_ARRAY object (described in Section 1.6.5) with the node ID values of nodes that have been added to the current updatable TopoMap object:

```
SELECT SDO_TOPO_MAP.GET_NODE_ADDITIONS FROM DUAL;
```

2.1.4 Process for Editing Using Cache Explicitly (PL/SQL API)

Figure 2–1 shows the recommended process for editing topology objects using the PL/SQL API and explicitly using a TopoMap object and its associated cache.

Figure 2–1 Editing Topologies Using the TopoMap Object Cache (PL/SQL API)


Create TopoMap object (CREATE_TOPO_MAP)
Load TopoMap object for update (LOAD_TOPO_MAP)
Perform editing operations (for example, add 1000 nodes)
Validate cache (VALIDATE_TOPO_MAP)
Rebuild indexes (CREATE_EDGE_INDEX and CREATE_FACE_INDEX) (for example, after each 100 added nodes)
Clear cache (CLEAR_TOPO_MAP)
Update topology (UPDATE_TOPO_MAP) (for example, after each 1000 added nodes)
Commit changes (COMMIT_TOPO_MAP)
Remove TopoMap object (DROP_TOPO_MAP)

As Figure 2–1 shows, the basic sequence is as follows:

1. Create the TopoMap object, using the SDO_TOPO_MAP.CREATE_TOPO_MAP procedure.

   This creates an in-memory cache for editing objects associated with the specified topology.

2. Load the entire topology or a rectangular window from the topology into the TopoMap object cache for update, using the SDO_TOPO_MAP.LOAD_TOPO_MAP procedure.

   You can specify that in-memory R-tree indexes be built on the edges and faces that are being loaded. These indexes use some memory resources and take some time to create and periodically rebuild; however, they significantly improve performance if you edit a large number of topology objects in the session. (They can also improve performance for queries that use a read-only TopoMap object.)

3. Perform a number of topology editing operations (for example, add 1000 nodes).

   Periodically, validate the cache by calling the SDO_TOPO_MAP.VALIDATE_TOPO_MAP function.

   You can rebuild existing in-memory R-tree indexes on edges and faces in the TopoMap object, or create new indexes if none exist, by using the SDO_TOPO_MAP.CREATE_EDGE_INDEX and SDO_TOPO_MAP.CREATE_FACE_INDEX procedures. For best index performance, these indexes should be rebuilt periodically when you are editing a large number of topology objects.

   If you want to discard edits made in the cache, call the SDO_TOPO_MAP.CLEAR_TOPO_MAP procedure. This procedure fails if there are any uncommitted updates; otherwise, it clears the data in the cache and sets the cache to be read-only.

4. Update the topology by calling the SDO_TOPO_MAP.UPDATE_TOPO_MAP procedure.

5. Repeat Steps 3 and 4 (editing objects, validating the cache, rebuilding the R-tree indexes, and updating the topology) as often as needed, until you have finished the topology editing operations.

6. Commit the topology changes by calling the SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure. (The SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure automatically performs the actions of the SDO_TOPO_MAP.UPDATE_TOPO_MAP procedure before it commits the changes.) After the commit operation, the cache is made read-only (that is, it is no longer

updatable). However, if you want to perform further editing operations using the same TopoMap object, you can load it again and use it (that is, repeat Steps 2 through 5, clearing the cache first if necessary).

To perform further editing operations, clear the TopoMap object cache by calling the SDO_TOPO_MAP.CLEAR_TOPO_MAP procedure, and then go to Step 2.

If you want to discard all uncommitted topology changes, you can call the SDO_TOPO_MAP.ROLLBACK_TOPO_MAP procedure at any time. After the rollback operation, the cache is cleared.

7. Remove the TopoMap object by calling the SDO_TOPO_MAP.DROP_TOPO_MAP procedure.

   This procedure removes the TopoMap object and frees any resources that it had used. (If you forget to drop the TopoMap object, it will automatically be dropped when the user session ends.) This procedure also rolls back any topology changes in the cache that have not been committed.

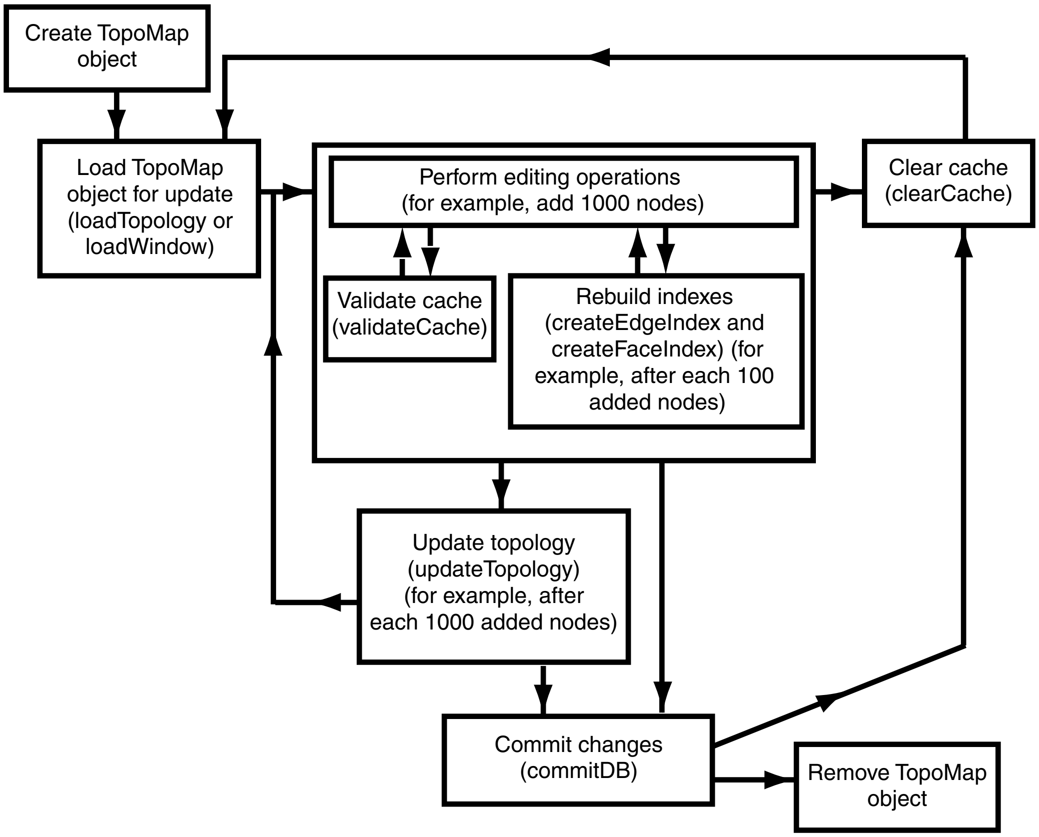
   If the application terminates abnormally, all uncommitted changes to the database will be discarded.

If you plan to perform a very large number of topology editing operations, you can divide the operations among several editing sessions, each of which performs Steps 1 through 7 in the preceding list.

2.1.5 Process for Editing Using the Java API

Figure 2–2 shows the recommended process for editing topology objects using the client-side Java API, which is introduced in Section 1.8.1 and is explained in the Javadoc-generated documentation. The Java API requires that you create and manage a TopoMap object and its associated cache.

Figure 2–2 Editing Topologies Using the TopoMap Object Cache (Java API)


Create TopoMap object
Load TopoMap object for update (loadTopology or loadWindow)
Perform editing operations (for example, add 1000 nodes)
Validate cache (validateCache)
Rebuild indexes (createEdgeIndex and createFaceIndex) (for example, after each 100 added nodes)
Clear cache (clearCache)
Update topology (updateTopology) (for example, after each 1000 added nodes)
Commit changes (commitDB)
Remove TopoMap object

As Figure 2–2 shows, the basic sequence is as follows:

1. Create the TopoMap object, using a constructor of the `TopoMap` class, specifying a topology and a database connection.

   This creates an in-memory cache for editing objects associated with the specified topology.

2. Load the entire topology or a rectangular window from the topology into the TopoMap object cache for update, using the `loadTopology` or `loadWindow` method of the `TopoMap` class.

You can specify that in-memory R-tree indexes be built on the edge and edge face that are being affected. These indexes use some memory resources and take some time to create and periodically rebuild; however, they significantly improve performance if you edit a large number of topology objects during the database connection.

3. Perform a number of topology editing operations (for example, add 1000 nodes), and update the topology by calling the `updateTopology` method of the `TopoMap` class.

   Periodically, validate the cache by calling the `validateCache` method of the `TopoMap` class.

   If you caused in-memory R-tree indexes to be created when you loaded the TopoMap object (in Step 2), you can periodically (for example, after each addition of 100 nodes) rebuild the indexes by calling the `createEdgeIndex` and `createFaceIndex` methods of the `TopoMap` class. For best index performance, these indexes should be rebuilt periodically when you are editing a large number of topology objects.

   If you do not want to update the topology but instead want to discard edits made in the cache since the last update, call the `clearCache` method of the `TopoMap` class. The `clearCache` method fails if there are any uncommitted updates; otherwise, it clears the data in the cache and sets the cache to be read-only.

4. Update the topology by calling the `updateTopology` method of the `TopoMap` class.

5. Repeat Steps 3 and 4 (editing objects, validating the cache, rebuilding the R-tree indexes, and updating the topology) as often as needed, until you have finished the topology editing operations.

6. Commit the topology changes by calling the `commitDB` method of the `TopoMap` class. (The `commitDB` method automatically calls the `updateTopology` method before it commits the changes.) After the commit operation, the cache is made read-only (that is, it is no longer updatable). However, if you want to perform further editing operations using the same TopoMap object, you can load it again and use it (that is, repeat Steps 2 through 5, clearing the cache first if necessary).

   To perform further editing operations, clear the TopoMap object cache by calling the `clearCache` method of the `TopoMap` class, and then go to Step 2.

If you want to discard all uncommitted topology changes, you can call the `rollbackDB` method of the `TopoMap` class at any time. After the rollback operation, the cache is cleared.

7. Remove the TopoMap object by setting the TopoMap object to null, which makes the object available for garbage collection and frees any resources that it had used. (If you forget to remove the TopoMap object, it will automatically be garbage collected when the application ends.)

   If the application terminates abnormally, all uncommitted changes to the database will be discarded.

If you plan to perform a very large number of topology editing operations, you can divide the operations among several editing sessions, each of which performs Steps 1 through 7 in the preceding list.

2.1.6 Error Handling for Topology Editing

This section discusses the following conditions:

- Input parameter errors
- All exceptions

2.1.6.1 Input Parameter Errors

When an SDO_TOPO_MAP PL/SQL subprogram or a public static method in the `TopoMap` Java class is called, it validates the values of the input parameters, and it uses or creates a TopoMap object to perform the editing or read-only operation. Whenever there is an input error, an `oracle.spatial.topo.TopoDataException` exception is thrown. Other errors may occur when the underlying TopoMap object performs an operation.

If the method is called from SQL or PL/SQL, the caller gets the following error message:

```
ORA-29532: Java call terminated by uncaught Java exception:
<specific error message text>
```

The following PL/SQL example shows how you can handle a `TopoDataException` exception:

```
DECLARE
  topo_data_error EXCEPTION;
  PRAGMA EXCEPTION_INIT(topo_data_error, -29532);
BEGIN
```

```
  sdo_topo_map.create_topo_map(null, null, 100, 100, 100);
EXCEPTION
  WHEN topo_data_error THEN
    DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

The preceding example generates the following output:

```
ORA-29532: Java call terminated by uncaught Java exception:
oracle.spatial.topo.TopoDataException: invalid TopoMap name
```

2.1.6.2 All Exceptions

The following actions are performed automatically when any exception occurs in a call to any of the following SDO_TOPO_MAP PL/SQL subprograms or their associated methods in the `TopoMap` Java class: SDO_TOPO_MAP.ADD_EDGE (`addEdge`), SDO_TOPO_MAP.ADD_ISOLATED_NODE (`addIsolatedNode`), SDO_TOPO_MAP.ADD_LOOP (`addLoop`), SDO_TOPO_MAP.ADD_NODE (`addNode`), SDO_TOPO_MAP.CHANGE_EDGE_COORDS (`changeEdgeCoords`), SDO_TOPO_MAP.MOVE_ISOLATED_NODE (`moveIsolatedNode`), SDO_TOPO_MAP.MOVE_NODE (`moveNode`), SDO_TOPO_MAP.MOVE_EDGE (`moveEdge`), SDO_TOPO_MAP.REMOVE_EDGE (`removeEdge`), SDO_TOPO_MAP.REMOVE_NODE (`removeNode`), and SDO_TOPO_MAP.UPDATE_TOPO_MAP (`updateTopology`).

- The transaction is rolled back.
- The TopoMap object cache is cleared.
- The TopoMap object is made read-only.

2.2 Performing Operations on Nodes

This section contains sections that describe the effects of adding, moving, and removing nodes, and that explain how to perform these operations using the PL/SQL API.

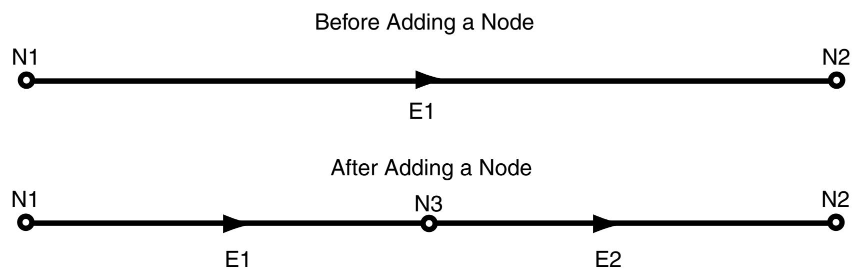
2.2.1 Adding a Node

Adding a non-isolated node adds the node to an edge at a point that is currently on the edge. This operation also splits the edge, causing the original edge to be divided into two edges. Spatial automatically adjusts the definition of the original edge and creates a new edge (assigning it an ID value that is unique among edges in the topology).

To add a non-isolated node, use the SDO_TOPO_MAP.ADD_NODE function. To add an isolated node, use the SDO_TOPO_MAP.ADD_ISOLATED_NODE function.

Figure 2–3 shows the addition of a node (N3) on edge E1.

Figure 2–3 Adding a Non-Isolated Node



As a result of the operation shown in Figure 2–3:

- Edge E1 is redefined to be between the original edge's start point and the point at the added node (N3).
- Edge E2 is created. Its start point is the point at node N3, and its end point is the end point of the original edge.

Any linear features that were defined on the original edge are automatically redefined to be on both resulting edges. For example, if a street named Main Street had been defined on the original edge E1 in Figure 2–3, then after the addition of node N3, Main Street would be defined on both edges E1 and E2.

2.2.2 Moving a Node

Moving a non-isolated node to a new position causes the ends of all edges that are attached to the node to move with the node. You must specify the vertices for all edges affected by the moving of the node; each point (start or end) that is attached to the node must have the same coordinates as the new location of the node, and the other end points (not the moved node) of each affected edge must remain the same.

To move a non-isolated node, use the SDO_TOPO_MAP.MOVE_NODE procedure. To move an isolated node, use the SDO_TOPO_MAP.MOVE_ISOLATED_NODE procedure.

Figure 2–4 shows the original topology before node N1 is moved.

Figure 2–4 Topology Before Moving a Non-Isolated Node

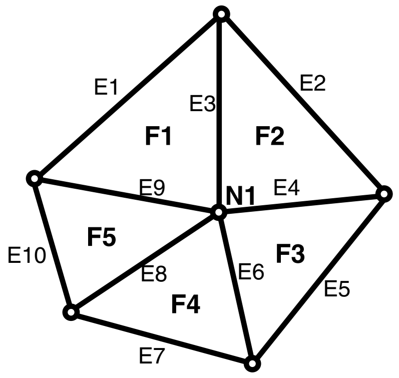
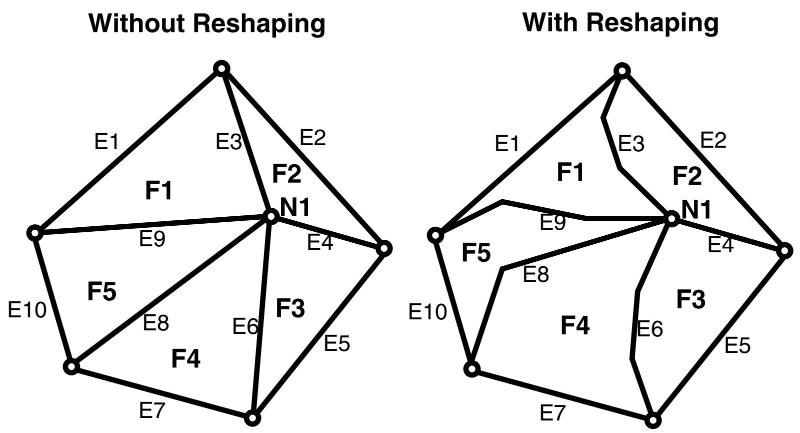


Figure 2–5 shows two cases of the original topology after node N1 is moved. In one case, no reshaping occurs; that is, all edges affected by the movement are specified as straight lines. In the other case, reshaping occurs; that is, one or more affected edges are specified as line segments with multiple vertices.

Figure 2–5 Topology After Moving a Non-Isolated Node



In both cases shown in Figure 2–5:

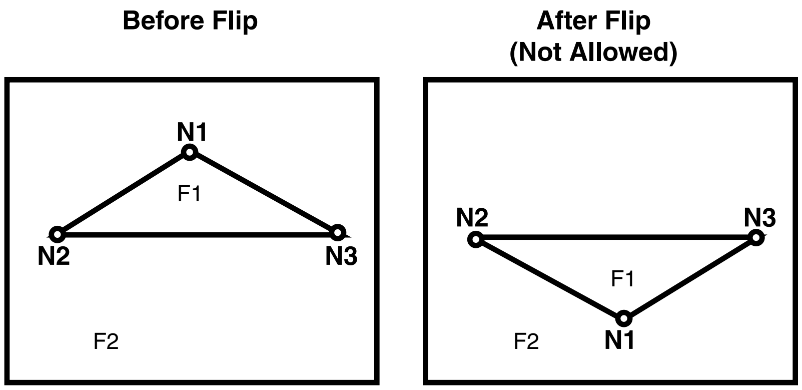
- The topology does not change. That is, the number of nodes, edges, and faces does not change, and the relationships among the nodes, edges, and faces do not change.
- All features defined on the nodes, edges, and faces retain their definitions.

Any isolated nodes and edges might remain in the same face or be moved to a different face as a result of a move operation on a non-isolated node. The SDO_TOPO_MAP.MOVE_NODE procedure has two output parameters, `moved_iso_nodes` and `moved_iso_edges`, that store the ID numbers of any isolated nodes and edges that were moved to a different face as a result of the operation.

A node cannot be moved if, as a result of the move, any of the following would happen:

- Any edges attached to the node would intersect any other edge. For example, assume that the original topology shown in Figure 2–5 had included another edge E20 that passed just above and to the right of node N1. If the movement of node N1 would cause edge E3, E4, E6, E8, or E9 to intersect edge E20, the move operation is not performed.
- The node would be moved to a face that does not currently bound the node. For example, if the movement of node N1 would place it outside the original topology shown in Figure 2–5, the move operation is not performed.
- The node would be moved to the opposite side of an isolated face. This is not allowed because the move would change the topology by changing one or more of the following: the relationship or ordering of edges around the face, and the left and right face for each edge. Figure 2–6 shows a node movement (flipping node N1 from one side of isolated face F1 to the other side) that would not be allowed.

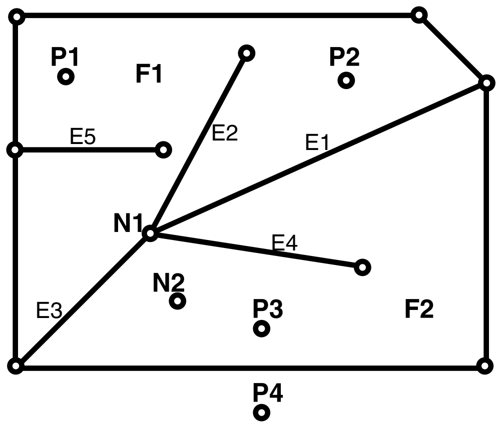
Figure 2–6 Node Move Is Not Allowed



2.2.2.1 Additional Examples of Allowed and Disallowed Node Moves

This section provides additional examples of node movement operations that are either allowed or not allowed. All refer to the topology shown in Figure 2–7.

Figure 2–7 Topology for Node Movement Examples



In the topology shown in Figure 2–7:

- Attempts will be made to move node N1 to points P1, P2, P3, and P4. (These points are locations but are not existing nodes.)

- The edges have no shape points, either before or after the move operation.
- New vertices are specified for the edges E1, E2, E3, and E4, but the ID values of the start and end points for the edges remain the same.

When the following node move operations are attempted using the topology shown in Figure 2–7, the following results occur:

- Moving node N1 to point P1: Not allowed, because one or more of the four attached edges would intersect edge E5. (Edge E3 would definitely intersect edge E5 if the move were allowed.)
- Moving node N1 to point P2: Allowed.
- Moving node N1 to point P3: Allowed. However, this operation causes the isolated node N2 to change from face F2 to face F1, and this might cause the application to want to roll back or disallow the movement of node N1. Similarly, if the movement of a node would cause any isolated edges or faces to change from one face to another, the application might want to roll back or disallow the node move operation.
- Moving node N1 to point P4: Not allowed, because the node must be moved to a point in a face that bounds the original (current) position of the node.

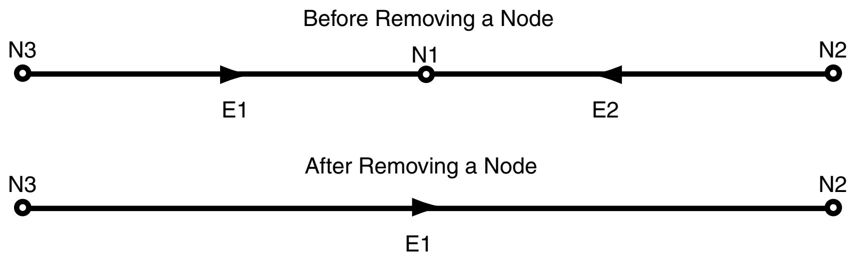
2.2.3 Removing a Node

Removing a non-isolated node deletes the node and merges the edges that were attached to the node into a single edge. (Spatial applies complex rules, which are not documented, to determine the ID value and direction of the resulting edge.)

To remove a non-isolated or isolated node, use the SDO_TOPO_MAP.REMOVE_NODE procedure.

Figure 2–8 shows the removal of a node (N1) that is attached to edges E1 and E2.

Figure 2–8 Removing a Non-Isolated Node



As a result of the operation shown in Figure 2–8:

- Edge E1 is redefined to consist of the line segments that had represented the original edges E1 and E2.
- Edge E2 is deleted.

Any linear features that were defined on both original edges are automatically redefined to be on the resulting edge. For example, if a street named Main Street had been defined on the original edges E1 and E2 in Figure 2–8, then after the removal of node N1, Main Street would be defined on edge E1.

A node cannot be removed if one or more of the following are true:

- A point feature is defined on the node. For example, if a point feature named Metropolitan Art Museum had been defined on node N1 in Figure 2–8, node N1 cannot be removed. Before you can remove the node in this case, you must remove the definition of any point features on the node.
- A linear feature defined on either original edge is not also defined on both edges. For example, if a linear feature named Main Street had been defined on edge E1 but not edge E2 in Figure 2–8, node N1 cannot be removed.

2.3 Performing Operations on Edges

This section contains sections that describe the effects of adding, moving, removing, and updating edges, and that explain how to perform these operations using the PL/SQL API.

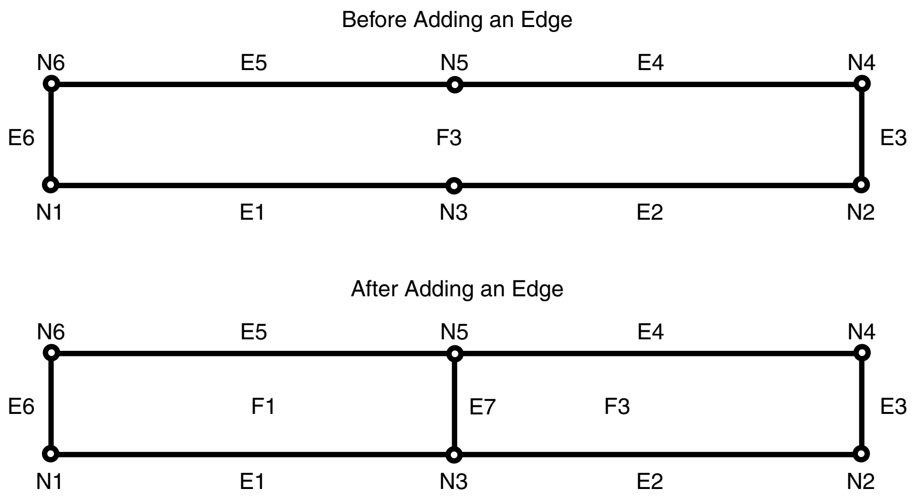
2.3.1 Adding an Edge

Adding a non-isolated edge adds the edge to a face. It also splits the face, causing the original face to be divided into two faces. Spatial automatically adjusts the definition of the original face and creates a new face (assigning it an ID value that is unique among faces in the topology).

To add an edge, use the SDO_TOPO_MAP.ADD_EDGE procedure. You must specify existing nodes as the start and end nodes of the added edge.

Figure 2–9 shows the addition of an edge (E7) between nodes N3 and N5 on face F3.

Figure 2–9 Adding a Non-Isolated Edge



As a result of the operation shown in Figure 2–9, face F3 is redefined to be two faces, F1 and F3. (Spatial applies complex rules, which are not documented, to determine the ID values of the resulting faces.)

Any polygon features that were defined on the original face are automatically redefined to be on both resulting faces. For example, if a park named Walden State Park had been defined on the original face F3 in Figure 2–9, then after the addition of edge E7, Walden State Park would be defined on both faces F1 and F3.

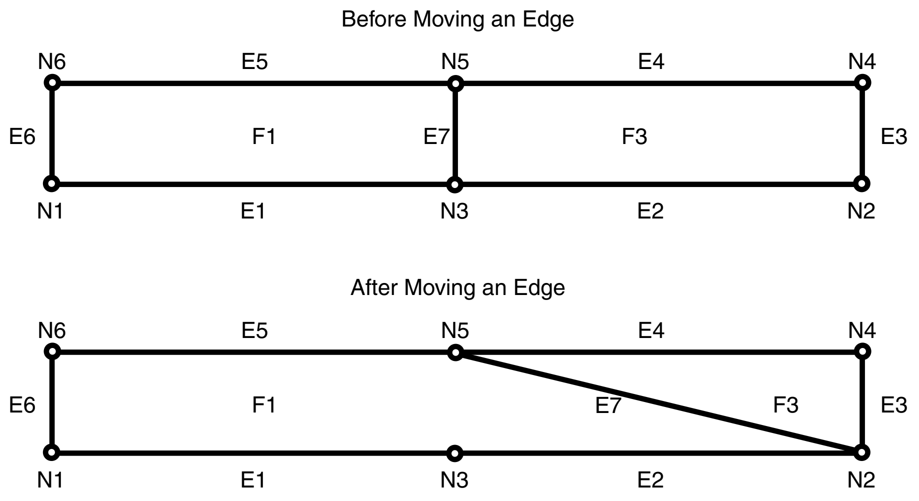
2.3.2 Moving an Edge

Moving a non-isolated edge keeps the start or end point of the edge in the same position and moves the other of those two points to another existing node position. You must specify the source node (location before the move of the node to be moved), the target node (location after the move of the node being moved), and the vertices for the moved edge.

To move an edge, use the SDO_TOPO_MAP.MOVE_EDGE procedure.

Figure 2–10 shows the movement of edge E7, which was originally between nodes N3 and N5, to be between nodes N2 and N5.

Figure 2–10 Moving a Non-Isolated Edge



As a result of the operation shown in Figure 2–10, faces F1 and F3 are automatically redefined to reflect the coordinates of their edges, including the new coordinates for edge E7.

Any isolated nodes and edges might remain in the same face or be moved to a different face as a result of a move operation on a non-isolated edge. The SDO_TOPO_MAP.MOVE_EDGE procedure has two output parameters, `moved_iso_nodes` and `moved_iso_edges`, that store the ID numbers of any isolated nodes and edges that were moved to a different face as a result of the operation.

An edge cannot be moved if, as a result of the move, any of the following would happen:

- The moved edge would intersect any other edge. For example, assume that the topology before the move, as shown in Figure 2–10, had included another edge (E10) that was between nodes N3 and N4. In this case, the movement of edge E7 would cause it to intersect edge E10, and therefore the move operation is not performed.
- The node would be moved to a face that does not currently bound the edge. For example, if the movement of edge E7 would place its terminating point at a node outside the faces shown in Figure 2–10 (F1 and F3), the move operation is not performed.

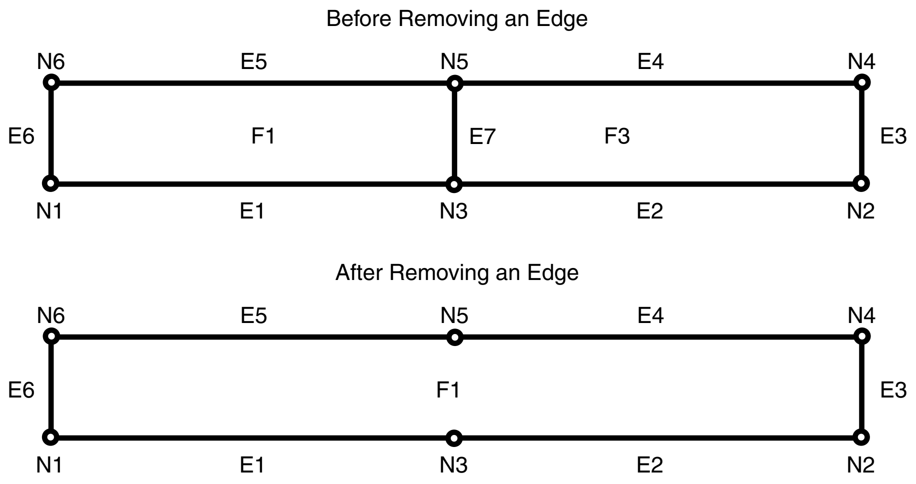
2.3.3 Removing an Edge

Removing a non-isolated edge deletes the edge and merges the faces that bounded the edge. (Spatial applies complex rules, which are not documented, to determine the ID value of the resulting face.)

To remove an edge, use the SDO_TOPO_MAP.REMOVE_EDGE procedure.

Figure 2–11 shows the removal of an edge (E7) that is bounded by faces F1 and F3.

Figure 2–11 Removing a Non-Isolated Edge



As a result of the operation shown in Figure 2–11:

- Face F1 is redefined to consist of the area of the original faces F1 and F3.
- Face F3 is deleted.
- The start and end nodes of the deleted edge (nodes N3 and N5) are not removed.

Any polygon features that were defined on both original faces are automatically redefined to be on the resulting face. For example, if a park named Adams Park had been defined on the original faces F1 and F3 in Figure 2–11, then after the removal of edge E7, Adams Park would be defined on face F1.

A non-isolated edge cannot be removed if one or more of the following are true:

- A linear feature is defined on the edge. For example, if a linear feature named Main Street had been defined on edge E7 in Figure 2–11, edge E7 cannot be removed. Before you can remove the edge in this case, you must remove the definition of any linear features on the edge.
- A polygon feature defined on either original face is not also defined on both faces. For example, if a linear feature named Adams Park had been defined on face F1 but not face F3 in Figure 2–11, edge E7 cannot be removed.

2.3.4 Updating an Edge

Updating an isolated edge means changing one or more coordinates of the edge, but without changing the start point and end point.

To update an edge, use the SDO_TOPO_MAP.CHANGE_EDGE_COORDS procedure.

An edge cannot be updated if, as a result of the operation, it would intersect any other edge. See the Usage Notes for the SDO_TOPO_MAP.CHANGE_EDGE_COORDS procedure for more information about updating an edge.

3 SDO_TOPO Package Subprograms

The MDSYS.SDO_TOPO package contains subprograms (functions and procedures) that constitute part of the PL/SQL application programming interface (API) for the Spatial topology data model. This package mainly contains subprograms for creating and managing topologies.

To use the subprograms in this chapter, you must understand the conceptual information about topology in Chapter 1.

Another package, SDO_TOPO_MAP, mainly contains subprograms related to editing topologies. Reference information for the SDO_TOPO_MAP package is in Chapter 4.

The rest of this chapter provides reference information on the SDO_TOPO subprograms, listed in alphabetical order.

SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER

Format

```
SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER(
    topology                  IN VARCHAR2,
    table_name                IN VARCHAR2,
    column_name               IN VARCHAR2,
    topo_geometry_layer_type  IN VARCHAR2,
    relation_table_storage    IN VARCHAR2 DEFAULT NULL,
    child_layer_id            IN NUMBER DEFAULT NULL);
```

Description

Adds a topology geometry layer to a topology.

Parameters

topology
Topology to which to add the topology geometry layer containing the topology geometries in the specified column. The topology must have been created using the SDO_TOPO.CREATE_TOPOLOGY procedure.

table_name
Name of the topology geometry layer table containing the column specified in `column_name`.

column_name
Name of the column (of type SDO_TOPO_GEOMETRY) containing the topology geometries in the topology geometry layer to be added to the topology.

topo_geometry_layer_type
Type of topology geometry layer: POINT, LINE, CURVE, or POLYGON.

relation_table_storage
Physical storage parameters used internally to create the <topology-name>_RELATION$ table (described in Section 1.5.4). Must be a valid string for use with the CREATE TABLE statement. For example: `TABLESPACE tbs_3 STORAGE`

(INITIAL 100K NEXT 200K). If you do not specify this parameter, the default physical storage values are used.

child_layer_id
Layer ID number of the child topology geometry layer for this layer, if the topology has a topology geometry layer hierarchy. (Topology geometry layer hierarchy is explained in Section 1.4.) If you do not specify this parameter and if the topology has a topology geometry layer hierarchy, the topology geometry layer is added to the lowest level (level 0) of the hierarchy.

If the topology does not have a topology geometry layer hierarchy, do not specify this parameter when adding any of the topology geometry layers.

Usage Notes

The first call to this procedure for a given topology creates the <topology-name>_RELATION$ table, which is described in Section 1.5.4.

An exception is raised if `topology`, `table_name`, or `column_name` does not exist, or if `topo_geometry_layer_type` is not one of the supported values.

Examples

The following example adds a topology geometry layer to the `CITY_DATA` topology. The topology geometry layer consists of polygon geometries in the FEATURE column of the LAND_PARCELS table. (The example refers to definitions and data from Section 1.11.)

```
EXECUTE SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('CITY_DATA', 'LAND_PARCELS', 'FEATURE',
'POLYGON');
```

SDO_TOPO.CREATE_TOPOLOGY

Format

```
SDO_TOPO.CREATE_TOPOLOGY(
    topology              IN VARCHAR2,
    tolerance             IN NUMBER,
    srid                  IN NUMBER DEFAULT NULL,
    node_table_storage    IN VARCHAR2 DEFAULT NULL,
    edge_table_storage    IN VARCHAR2 DEFAULT NULL,
    face_table_storage    IN VARCHAR2 DEFAULT NULL,
    history_table_storage IN VARCHAR2 DEFAULT NULL);
```

Description

Creates a topology.

Parameters

topology
Name of the topology to be created. Must not exceed 20 characters.

tolerance
Tolerance value associated with topology geometries in the topology. (Tolerance is explained in Chapter 1 of *Oracle Spatial User's Guide and Reference*.) Oracle Spatial uses the tolerance value in building R-tree indexes on the node, edge, and face tables; the value is also used for any spatial queries that use these tables.

srid
Coordinate system (spatial reference system) associated with all topology geometry layers in the topology. The default is null: no coordinate system is associated; otherwise, it must be a value from the SRID column of the MDSYS.CS_SRS table (described in *Oracle Spatial User's Guide and Reference*).

node_table_storage
Physical storage parameters used internally to create the <topology-name>_NODE$ table (described in Section 1.5.2). Must be a valid string for use with the CREATE

TABLE statement. For example: `TABLESPACE tbs_3 STORAGE (INITIAL 100K NEXT 200K)`. If you do not specify this parameter, the default physical storage values are used.

edge_table_storage
Physical storage parameters used internally to create the <topology-name>_EDGE$ table (described in Section 1.5.1). Must be a valid string for use with the CREATE TABLE statement. For example: `TABLESPACE tbs_3 STORAGE (INITIAL 100K NEXT 200K)`. If you do not specify this parameter, the default physical storage values are used.

face_table_storage
Physical storage parameters used internally to create the <topology-name>_FACE$ table (described in Section 1.5.3). Must be a valid string for use with the CREATE TABLE statement. For example: `TABLESPACE tbs_3 STORAGE (INITIAL 100K NEXT 200K)`. If you do not specify this parameter, the default physical storage values are used.

history_table_storage
Physical storage parameters used internally to create the <topology-name>_HISTORY$ table (described in Section 1.5.5. Must be a valid string for use with the CREATE TABLE statement. For example: `TABLESPACE tbs_3 STORAGE (INITIAL 100K NEXT 200K)`. If you do not specify this parameter, the default physical storage values are used.

Usage Notes

This procedure creates the <topology-name>_EDGE$, <topology-name>_NODE$, <topology-name>_FACE$, and <topology-name>_HISTORY$ tables, which are described in Section 1.5. This procedure also creates the metadata for the topology.

An exception is raised if the topology already exists.

Examples

The following example creates a topology named `CITY_DATA`. The spatial geometries in this topology have a tolerance value of 0.5 and use the `WGS 84` coordinate system (longitude and latitude, SRID value 8307). (The example refers to definitions and data from Section 1.11.)

```
EXECUTE SDO_TOPO.CREATE_TOPOLOGY('CITY_DATA', 0.5, 8307);
```

SDO_TOPO.DELETE_TOPO_GEOMETRY_LAYER

Format

```
SDO_TOPO.DELETE_TOPO_GEOMETRY_LAYER(
     topology       IN VARCHAR2,
     table_name     IN VARCHAR2,
     column_name  IN VARCHAR2);
```

Description

Deletes a topology geometry layer from a topology.

Parameters

topology
Topology from which to delete the topology geometry layer containing the topology geometries in the specified column. The topology must have been created using the SDO_TOPO.CREATE_TOPOLOGY procedure.

table_name
Name of the table containing the column specified in `column_name`.

column_name
Name of the column containing the topology geometries in the topology geometry layer to be deleted from the topology.

Usage Notes

This procedure deletes data associated with the specified topology geometry layer from the <topology-name>_RELATION$ table (described in Section 1.5.4). If this procedure is deleting the only remaining topology geometry layer from the topology, it also deletes the <topology-name>_RELATION$ table.

Examples

The following example deletes the topology geometry layer that is based on the geometries in the FEATURE column of the LAND_PARCELS table from the topology named `CITY_DATA`. (The example refers to definitions and data from Section 1.11.)

```
EXECUTE SDO_TOPO.DELETE_TOPO_GEOMETRY_LAYER('CITY_DATA', 'LAND_PARCELS',
'FEATURE');
```

SDO_TOPO.DROP_TOPOLOGY

Format

```
SDO_TOPO.DROP_TOPOLOGY(
    topology  IN VARCHAR2);
```

Description

Deletes a topology.

Parameters

topology
Name of the topology to be deleted. The topology must have been created using the SDO_TOPO.CREATE_TOPOLOGY procedure.

Usage Notes

This procedure deletes the <topology-name>_EDGE$, <topology-name>_NODE$, <topology-name>_FACE$, and <topology-name>_HISTORY$ tables (described in Section 1.5).

An exception is raised if the topology contains any topology geometries from any topology geometry layers. If you encounter this exception, delete all topology geometry layers in the topology using the SDO_TOPO.DELETE_TOPO_GEOMETRY_LAYER procedure for each topology geometry layer, and then drop the topology.

Examples

The following example drops the topology named CITY_DATA. (The example refers to definitions and data from Section 1.11.)

```
EXECUTE SDO_TOPO.DROP_TOPOLOGY('CITY_DATA');
```

SDO_TOPO.GET_FACE_BOUNDARY

Format

```
SDO_TOPO.GET_FACE_BOUNDARY(
    topology   IN VARCHAR2,
    face_id    IN NUMBER,
    all_edges  IN VARCHAR2 DEFAULT 'FALSE'
    ) RETURN SDO_LIST_TYPE;
```

Description

Returns a list of the signed ID numbers of the edges for the specified face.

Parameters

topology
Name of the topology that contains the face. Must not exceed 20 characters.

face_id
Face ID value of the face.

all_edges
`TRUE` includes all edges in the face, including isolated edges and edges that intersect a point on an edge on the boundary of the face; `FALSE` (the default) includes only edges that constitute the boundary of the face. (See the examples for this function.)

Usage Notes

None.

Examples

The following examples return the ID numbers of the edges for the face whose face ID value is 1. The first example accepts the default value of `'FALSE'` for the `all_edges` parameter. The second example specifies `'TRUE'` for `all_edges`, and the list includes the ID numbers of the boundary edge and the two isolated edges on the face. (The examples refer to definitions and data from Section 1.11.)

```
-- Get the boundary of face with face_id 1.
SELECT SDO_TOPO.GET_FACE_BOUNDARY('CITY_DATA', 1) FROM DUAL;

SDO_TOPO.GET_FACE_BOUNDARY('CITY_DATA',1)
--------------------------------------------------------------------------------
SDO_LIST_TYPE(1)

-- Specify 'TRUE' for the all_edges parameter.
SELECT SDO_TOPO.GET_FACE_BOUNDARY('CITY_DATA', 1, 'TRUE') FROM DUAL;

SDO_TOPO.GET_FACE_BOUNDARY('CITY_DATA',1,'TRUE')
--------------------------------------------------------------------------------
SDO_LIST_TYPE(1, -26, 25)
```

SDO_TOPO.GET_TOPO_OBJECTS

Format

```
SDO_TOPO.GET_TOPO_OBJECTS(
    topology    IN VARCHAR2,
    geometry  IN SDO_GEOMETRY
    ) RETURN SDO_TOPO_OBJECT_ARRAY;
```

or

```
SDO_TOPO.GET_TOPO_OBJECTS(
    topology                  IN VARCHAR2,
    topo_geometry_layer_id  IN NUMBER,
    topo_geometry_id         IN NUMBER
    ) RETURN SDO_TOPO_OBJECT_ARRAY;
```

Description

Returns an array of SDO_TOPO_OBJECT objects that interact with a specified geometry object or topology geometry object.

Parameters

topology
Name of the topology that contains the face and the point. Must not exceed 20 characters.

geometry
Geometry object to be checked.

topo_geometry_layer_id
ID number of the topology geometry layer that contains the topology geometry object to be checked.

topo_geometry_id
ID number of the topology geometry object to be checked.

Usage Notes

The SDO_TOPO_OBJECT_ARRAY data type is described in Section 1.6.2.1.

Examples

The following example returns the topology geometry objects that interact with land parcel P2 in the `CITY_DATA` topology. (The example refers to definitions and data from Section 1.11.)

```
-- CITY_DATA layer, land parcels (topo_geometry_ layer_id = 1),
-- parcel P2 (topo_geometry_id = 2)
SELECT SDO_TOPO.GET_TOPO_OBJECTS('CITY_DATA', 1, 2) FROM DUAL;

SDO_TOPO.GET_TOPO_OBJECTS('CITY_DATA',1,2)(TOPO_ID, TOPO_TYPE)
--------------------------------------------------------------------------------
SDO_TOPO_OBJECT_ARRAY(SDO_TOPO_OBJECT(9, 1), SDO_TOPO_OBJECT(10, 1), SDO_TOPO_OB
JECT(13, 1), SDO_TOPO_OBJECT(14, 1), SDO_TOPO_OBJECT(17, 1), SDO_TOPO_OBJECT(18,
 1), SDO_TOPO_OBJECT(6, 2), SDO_TOPO_OBJECT(7, 2), SDO_TOPO_OBJECT(8, 2), SDO_TO
PO_OBJECT(9, 2), SDO_TOPO_OBJECT(10, 2), SDO_TOPO_OBJECT(11, 2), SDO_TOPO_OBJECT
(12, 2), SDO_TOPO_OBJECT(13, 2), SDO_TOPO_OBJECT(14, 2), SDO_TOPO_OBJECT(17, 2),
 SDO_TOPO_OBJECT(18, 2), SDO_TOPO_OBJECT(19, 2), SDO_TOPO_OBJECT(20, 2), SDO_TOP
O_OBJECT(-6, 2), SDO_TOPO_OBJECT(-7, 2), SDO_TOPO_OBJECT(-8, 2), SDO_TOPO_OBJECT
(-9, 2), SDO_TOPO_OBJECT(-10, 2), SDO_TOPO_OBJECT(-11, 2), SDO_TOPO_OBJECT(-12,
2), SDO_TOPO_OBJECT(-13, 2), SDO_TOPO_OBJECT(-14, 2), SDO_TOPO_OBJECT(-17, 2), S
DO_TOPO_OBJECT(-18, 2), SDO_TOPO_OBJECT(-19, 2), SDO_TOPO_OBJECT(-20, 2), SDO_TO
PO_OBJECT(-1, 3), SDO_TOPO_OBJECT(3, 3), SDO_TOPO_OBJECT(4, 3), SDO_TOPO_OBJECT(
5, 3), SDO_TOPO_OBJECT(6, 3), SDO_TOPO_OBJECT(7, 3), SDO_TOPO_OBJECT(8, 3))
```

SDO_TOPO.INITIALIZE_METADATA

Format

```
SDO_TOPO.INITIALIZE_METADATA(
     topology  IN VARCHAR2);
```

Description

Initializes the topology metadata: sets sequence information for the node, edge, and face tables, and creates (or re-creates) required indexes on these tables.

Parameters

topology
Name of the topology for which to initialize the sequences. The topology must have been created using the SDO_TOPO.CREATE_TOPOLOGY procedure.

Usage Notes

You should run this procedure after loading data into the node, edge, or face tables, to initialize the sequences for these tables with the highest ID values stored in those tables. This ensures that no attempt is made to reuse the unique ID values in these tables. (The node, edge, and face tables are described in Section 1.5.)

This procedure creates spatial indexes on the geometry or MBR geometry columns in the node, edge, and face tables. If the indexes were dropped before a bulk load operation, running this procedure after the bulk load will re-create these indexes.

Examples

The following example initializes the metadata for the topology named `CITY_DATA`. (The example refers to definitions and data from Section 1.11.)

```
EXECUTE SDO_TOPO.INITIALIZE_METADATA('CITY_DATA');
```

4

SDO_TOPO_MAP Package Subprograms

The MDSYS.SDO_TOPO_MAP package contains subprograms (functions and procedures) that constitute part of the PL/SQL application programming interface (API) for the Spatial topology data model. This package contains subprograms related to editing topologies. These subprograms use a TopoMap object, either one that you previously created or that Spatial creates implicitly.

To use the subprograms in this chapter, you must understand the conceptual information about topology in Chapter 1, as well as the information about editing topologies in Chapter 2.

The rest of this chapter provides reference information on the SDO_TOPO_MAP subprograms, listed in alphabetical order.

SDO_TOPO_MAP.ADD_EDGE

Format

```
SDO_TOPO_MAP.ADD_EDGE(
     topology    IN VARCHAR2,
     node_id1    IN NUMBER,
     node_id2    IN NUMBER,
     geom        IN SDO_GEOMETRY
     ) RETURN NUMBER;
```

Description

Adds an edge to a topology, and returns the edge ID of the added edge.

Parameters

topology
Name of the topology to which to add the edge, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

node_id1
Node ID of the start node for the edge to be added.

node_id2
Node ID of the end node for the edge to be added.

geom
SDO_GEOMETRY object (line or contiguous line string geometry) representing the edge to be added.

Usage Notes

Spatial automatically assigns an edge ID to the added edge and inserts the appropriate entry in the <topology-name>_EDGE$ table. If the addition of the edge affects the face table information, Spatial automatically updates the appropriate entries in the <topology-name>_FACE$ table.

If `node_id1` and `node_id2` are the same value, a loop edge is created.

For information about adding and deleting nodes and edges, see Chapter 2.

This function is equivalent to using the `addEdge` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example adds an edge connecting node N3 to node N4 in the current updatable TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.ADD_EDGE(null, 3, 4,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(25,35, 20,37)))
  INTO :res_number;

Call completed.

SQL> PRINT res_number;

RES_NUMBER
----------
        29
```

SDO_TOPO_MAP.ADD_ISOLATED_NODE

Format

```
SDO_TOPO_MAP.ADD_ISOLATED_NODE(
     topology     IN VARCHAR2,
     face_id      IN NUMBER,
     point        IN SDO_GEOMETRY
     ) RETURN NUMBER;
```

or

```
SDO_TOPO_MAP.ADD_ISOLATED_NODE(
     topology     IN VARCHAR2,
     point        IN SDO_GEOMETRY
     ) RETURN NUMBER;
```

Description

Adds an isolated node (that is, an island node) to a topology, and returns the node ID of the added isolated node.

Parameters

topology
Name of the topology to which to add the isolated node, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

face_id
Face ID of the face on which the isolated node is to be added. (An exception is raised if the specified point is not on the specified face.)

point
SDO_GEOMETRY object (point geometry) representing the isolated node to be added.

Usage Notes

Spatial automatically assigns a node ID to the added node and inserts the appropriate entry in the <topology-name>_NODE$ table. Spatial also updates the <topology-name>_FACE$ table to include an entry for the added isolated node.

If you know the ID of the face on which the isolated node is to be added, you can specify the `face_id` parameter. If you specify this parameter, there are two benefits:

- Validation: The function checks to see if the point is on the specified face, and raises an exception if it is not. Otherwise, the function checks to see if the point is on any face in the topology, and raises an exception if it is not.
- Performance: The function checks only if the point is on the specified face. Otherwise, it checks potentially all faces in the topology to see if the point is on any face.

To add a non-isolated node, use the SDO_TOPO_MAP.ADD_NODE function.

For information about adding and deleting nodes and edges, see Chapter 2.

This function is equivalent to using the `addIsolatedNode` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example adds an isolated node to the right of isolated node N4 on face F2, and it returns the node ID of the added node. It uses the current updatable TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
DECLARE
  result_num NUMBER;
BEGIN
result_num := SDO_TOPO_MAP.ADD_ISOLATED_NODE(null, 2,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(22,37,NULL), NULL, NULL));
DBMS_OUTPUT.PUT_LINE('Result = ' || result_num);
END;
/
Result = 24

PL/SQL procedure successfully completed.
```

SDO_TOPO_MAP.ADD_LOOP

Format

```
SDO_TOPO_MAP.ADD_LOOP(
     topology    IN VARCHAR2,
     node_id     IN NUMBER,
     geom        IN SDO_GEOMETRY
     ) RETURN NUMBER;
```

Description

Adds an edge that loops and connects to the same node, and returns the edge ID of the added edge.

Parameters

topology
Name of the topology to which to add the edge, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

node_id
Node ID of the node to which to add the edge that will start and end at this node.

geom
SDO_GEOMETRY object (line string geometry) representing the edge to be added. The start and end points of the line string must be the same point representing `node_id`.

Usage Notes

This function creates a new edge, as well as a new face consisting of the interior of the loop. If the edge is added at an isolated node, the edge is an isolated edge. Spatial automatically updates the <topology-name>_EDGE$ and <topology-name>_FACE$ tables as needed.

For information about adding and deleting nodes and edges, see Chapter 2.

This function is equivalent to using the `addLoop` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example adds an edge loop starting and ending at node N4, and it returns the edge ID of the added edge. It uses the current updatable TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.ADD_LOOP(null, 4,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(20,37, 20,39, 25,39, 20,37)))
  INTO :res_number;

Call completed.

SQL> PRINT res_number;

RES_NUMBER
----------
        30
```

SDO_TOPO_MAP.ADD_NODE

Format

```
SDO_TOPO_MAP.ADD_NODE(
    topology           IN VARCHAR2,
    edge_id            IN NUMBER,
    point              IN SDO_GEOMETRY,
    coord_index        IN NUMBER,
    is_new_shape_point IN VARCHAR2
    ) RETURN NUMBER;
```

Description

Adds a non-isolated node to a topology to split an existing edge, and returns the node ID of the added node.

Parameters

topology
Name of the topology to which to add the node, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

edge_id
Edge ID of the edge on which the node is to be added.

point
SDO_GEOMETRY object (point geometry) representing the node to be added. The point must be an existing shape point or be on the line segment connecting two consecutive shape points.

coord_index
The index (position) of the array position in the edge coordinate array on or after which the node is to be added. Each vertex (node or shape point) has a position in the edge coordinate array. The start point (node) is index (position) 0, the first point after the start point is 1, and so on. (However, the `coord_index` value cannot be the index of the last vertex.) For example, if the edge coordinates are (2,2, 5,2, 8,3) the index of the second vertex (5,2) is 1.

is_new_shape_point
TRUE if the added node is to be a new shape point following the indexed vertex (coord_index value) of the edge; FALSE if the added node is exactly on the indexed vertex.

A value of TRUE lets you add a node at a new point, breaking an edge segment at the coordinates specified in the point parameter. A value of FALSE causes the coordinates in the point parameter to be ignored, and causes the node to be added at the existing shape point associated with the coord_index value.

Usage Notes

Spatial automatically assigns a node ID to the added node and inserts the appropriate entry in the <topology-name>_NODE$ table. Spatial also creates a new edge and inserts the appropriate entry in the <topology-name>_EDGE$ table.

To add an isolated node (that is, an island node), use the SDO_TOPO_MAP.ADD_ISOLATED_NODE function.

For information about adding and deleting nodes and edges, see Chapter 2.

This function is equivalent to using the addNode method of the TopoMap class of the client-side Java API (described in Section 1.8.1).

Examples

The following example adds a non-isolated node to the right of node N2 on edge E2, and it returns the node ID of the added node. It uses the current updatable TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
DECLARE
  result_num NUMBER;
BEGIN
result_num := SDO_TOPO_MAP.ADD_NODE(null, 2,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(27,30,NULL), NULL, NULL),
  0, 'TRUE');
DBMS_OUTPUT.PUT_LINE('Result = ' || result_num);
END;
/
Result = 26

PL/SQL procedure successfully completed.
```

SDO_TOPO_MAP.CHANGE_EDGE_COORDS

Format

```
SDO_TOPO_MAP.CHANGE_EDGE_COORDS(
    topology    IN VARCHAR2,
    edge_id     IN NUMBER,
    geom        IN SDO_GEOMETRY);
```

or

```
SDO_TOPO_MAP.CHANGE_EDGE_COORDS(
    topology          IN VARCHAR2,
    edge_id           IN NUMBER,
    geom              IN SDO_GEOMETRY,
    moved_iso_nodes  OUT SDO_NUMBER_ARRAY,
    moved_iso_edges  OUT SDO_NUMBER_ARRAY,
    allow_iso_moves   IN VARCHAR2);
```

Description

Changes the coordinates and related information about an edge.

Parameters

topology
Name of the topology containing the edge, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

edge_id
Edge ID of the edge whose coordinates are to be changed.

geom
SDO_GEOMETRY object (line or contiguous line string geometry) representing the modified edge. The start and end points of the modified edge must be the same as for the original edge.

moved_iso_nodes
Output parameter in which, if the `allow_iso_moves` parameter value is `TRUE`, Spatial stores the node ID values of any isolated nodes that have moved to a different face as a result of this procedure. If the `allow_iso_moves` parameter value is `FALSE`, Spatial stores the node ID values of any isolated nodes that did not move but that would have moved to a different face if the `allow_iso_moves` parameter value had been `TRUE`.

moved_iso_edges
Output parameter in which, if the `allow_iso_moves` parameter value is `TRUE`, Spatial stores the edge ID values of any isolated edges that have moved to a different face as a result of this procedure. If the `allow_iso_moves` parameter value is `FALSE`, Spatial stores the edge ID values of any isolated edges that did not move but that would have moved to a different face if the `allow_iso_moves` parameter value had been `TRUE`.

allow_iso_moves
`TRUE` causes Spatial to allow an edge coordinates change operation that would cause any isolated nodes or edges to be in a different face, and to adjust the containing face information for such isolated nodes and edges; `FALSE` causes Spatial not to allow an edge coordinates change operation that would cause any isolated nodes or edges to be in a different face.

If you use the format that does not include the `allow_iso_moves` parameter, Spatial allows edge move operations that would cause any isolated nodes or edges to be in a different face, and it adjusts the containing face information for such isolated nodes and edges.

Usage Notes

If this procedure modifies a boundary between faces, Spatial automatically performs the following operations and updates the topology data model tables as needed: reassigning island nodes and faces, and adjusting the MBRs of the faces on both sides.

This procedure modifies the information about the specified edge in the <topology-name>_EDGE$ table (described in Section 1.5.1).

You cannot use this procedure to change the start point or the end point, or both, of the specified edge. To do any of these operations, you must delete the edge, delete the node or nodes for the start or end point (or both) to be changed, add the necessary new node or nodes, and add the edge.

For information about editing topology objects, see Chapter 2.

This procedure is equivalent to using the `changeEdgeCoords` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example changes the coordinates of edge E1. (It changes only the third point, from 16,38 to 16,39.) It uses the current updatable TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.CHANGE_EDGE_COORDS(null, 1,
  SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(8,30, 16,30, 16,39, 3,38, 3,30, 8,30)));
```

SDO_TOPO_MAP.CLEAR_TOPO_MAP

Format

```
SDO_TOPO_MAP.CLEAR_TOPO_MAP(
     topo_map  IN VARCHAR2);
```

Description

Clears all objects and changes in the cache associated with a TopoMap object.

Parameters

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

Usage Notes

If the TopoMap object is updatable, this procedure changes it to be read-only.

For information about using an in-memory cache to edit topology objects, see Section 2.1.

Contrast this procedure with the SDO_TOPO_MAP.UPDATE_TOPO_MAP procedure, which applies the changes in the cache associated with the TopoMap object to the topology. You cannot call the SDO_TOPO_MAP.CLEAR_TOPO_MAP procedure if you previously used the SDO_TOPO_MAP.UPDATE_TOPO_MAP procedure on the specified TopoMap object.

This procedure is equivalent to using the `clearCache` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example clears the cache associated with the TopoMap object named `CITY_DATA_TOPOMAP`, which is associated with the topology named `CITY_DATA`. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.CLEAR_TOPO_MAP('CITY_DATA_TOPOMAP');
```

SDO_TOPO_MAP.COMMIT_TOPO_MAP

Format

```
SDO_TOPO_MAP.COMMIT_TOPO_MAP;
```

Description

Updates the topology to reflect changes made to the current updatable TopoMap object, commits all changes to the database, and makes the TopoMap object read-only.

Parameters

None.

Usage Notes

Use this procedure when you are finished with a batch of edits to a topology and you want to commit all changes to the database. After the commit operation completes, you cannot edit the TopoMap object. To make further edits to the topology, you must either clear the cache (using the SDO_TOPO_MAP.CLEAR_TOPO_MAP procedure) or create a new TopoMap object (using the SDO_TOPO_MAP.CREATE_TOPO_MAP procedure), and then load the topology into the TopoMap object for update (using the SDO_TOPO_MAP.LOAD_TOPO_MAP function).

Contrast this procedure with the SDO_TOPO_MAP.UPDATE_TOPO_MAP procedure, which leaves the TopoMap object available for editing operations and which does not perform a commit operation (and thus does not end the database transaction).

To roll back all TopoMap object changes, use the SDO_TOPO_MAP.ROLLBACK_TOPO_MAP procedure.

For information about using an in-memory cache to edit topology objects, see Section 2.1.

This procedure is equivalent to using the `commitDB` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example commits to the database all changes to the current updatable TopoMap object, and prevents further editing of the TopoMap object.

```
EXECUTE SDO_TOPO_MAP.COMMIT_TOPO_MAP;
```

SDO_TOPO_MAP.CREATE_EDGE_INDEX

Format

```
SDO_TOPO_MAP.CREATE_EDGE_INDEX(
     topo_map  IN VARCHAR2);
```

Description

Creates an internal R-tree index (or rebuilds the index if one already exists) on the edges in the cache associated with a TopoMap object.

Parameters

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

Usage Notes

You can cause Spatial to create in-memory R-tree indexes to be built on the edges and faces in the specified TopoMap object. These indexes use some memory resources and take some time to create; however, they significantly improve performance if you edit a large number of topology objects in the session. They can also improve performance for queries that use a read-only TopoMap object. If the TopoMap object is updatable and if you are performing many editing operations, you should probably rebuild the indexes periodically; however, if the TopoMap object will not be updated, create the indexes when or after loading the read-only topoMap object or after calling the SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure.

Compare this procedure with the SDO_TOPO_MAP.CREATE_FACE_INDEX procedure, which creates an internal R-tree index (or rebuilds the index if one already exists) on the faces in the cache associated with a TopoMap object.

This procedure is equivalent to using the `createEdgeIndex` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example creates an internal R-tree index (or rebuilds the index if one already exists) on the edges in the cache associated with the TopoMap object named

CITY_DATA_TOPOMAP, which is associated with the topology named CITY_DATA. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.CREATE_EDGE_INDEX('CITY_DATA_TOPOMAP');
```

SDO_TOPO_MAP.CREATE_FACE_INDEX

Format

```
SDO_TOPO_MAP.CREATE_FACE_INDEX(
     topo_map  IN VARCHAR2);
```

Description

Creates an internal R-tree index (or rebuilds the index if one already exists) on the faces in the cache associated with a TopoMap object.

Parameters

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

Usage Notes

You can cause Spatial to create in-memory R-tree indexes to be built on the edges and faces in the specified TopoMap object. These indexes use some memory resources and take some time to create; however, they significantly improve performance if you edit a large number of topology objects in the session. They can also improve performance for queries that use a read-only TopoMap object. If the TopoMap object is updatable and if you are performing many editing operations, you should probably rebuild the indexes periodically; however, if the TopoMap object will not be updated, create the indexes when or after loading the read-only topoMap object or after calling the SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure.

Compare this procedure with the SDO_TOPO_MAP.CREATE_EDGE_INDEX procedure, which creates an internal R-tree index (or rebuilds the index if one already exists) on the edges in the cache associated with a TopoMap object.

This procedure is equivalent to using the `createFaceIndex` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example creates an internal R-tree index (or rebuilds the index if one already exists) on the faces in the cache associated with the TopoMap object named

CITY_DATA_TOPOMAP, which is associated with the topology named CITY_DATA. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.CREATE_FACE_INDEX('CITY_DATA_TOPOMAP');
```

SDO_TOPO_MAP.CREATE_TOPO_MAP

Format

```
SDO_TOPO_MAP.CREATE_TOPO_MAP(
    topology          IN VARCHAR2,
    topo_map          IN VARCHAR2,
    number_of_edges   IN NUMBER DEFAULT 100,
    number_of_nodes   IN NUMBER DEFAULT 80,
    number_of_faces   IN NUMBER DEFAULT 30);
```

Description

Creates a TopoMap object cache associated with an existing topology.

Parameters

topology
Name of the topology. Must not exceed 20 characters.

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

number_of_edges
An estimate of the maximum number of edges that will be in the TopoMap object at any given time. If you do not specify this parameter, a default value of 100 is used.

number_of_nodes
An estimate of the maximum number of nodes that will be in the TopoMap object at any given time. If you do not specify this parameter, a default value of 80 is used.

number_of_faces
An estimate of the maximum number of faces that will be in the TopoMap object at any given time. If you do not specify this parameter, a default value of 30 is used.

Usage Notes

The `number_of_edges`, `number_of_nodes`, and `number_of_faces` parameters let you improve the performance and memory usage of the procedure when you have a good idea of the approximate number of edges, nodes, or faces (or any combination) that will be placed in the cache associated with the specified TopoMap object. Spatial initially allocates memory cache for the specified or default number of objects of each type, and incrementally increases the allocation later if more objects need to be accommodated.

You can create more than one TopoMap object in a user session; however, there can be no more than one updatable TopoMap object at any given time in a user session.

For information about using an in-memory cache to edit topology objects, see Section 2.1.

Using this procedure is equivalent to calling the constructor of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example creates a TopoMap object named `CITY_DATA_TOPOMAP` and its associated cache, and it associates the TopoMap object with the topology named `CITY_DATA`. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.CREATE_TOPO_MAP('CITY_DATA', 'CITY_DATA_TOPOMAP');
```

SDO_TOPO_MAP.DROP_TOPO_MAP

Format

```
SDO_TOPO_MAP.DROP_TOPO_MAP(
     topo_map  IN VARCHAR2);
```

Description

Deletes a TopoMap object from the current user session.

Parameters

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

Usage Notes

This procedure rolls back any uncommitted changes if the TopoMap object is updatable (that is, performs the equivalent of an SDO_TOPO_MAP.ROLLBACK_TOPO_MAP operation). It clears the cache associated with the TopoMap object, and removes the TopoMap object from the session.

For information about using an in-memory cache to edit topology objects, see Section 2.1.

Using this procedure is equivalent to setting the variable of the TopoMap object to a null value in a client-side Java application. (The client-side Java API is described in Section 1.8.1.)

Examples

The following example drops the TopoMap object named `CITY_DATA_TOPOMAP`. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.DROP_TOPO_MAP('CITY_DATA_TOPOMAP');
```

SDO_TOPO_MAP.GET_CONTAINING_FACE

Format

```
SDO_TOPO_MAP.GET_CONTAINING_FACE(
    topology    IN VARCHAR2,
    topo_map  IN VARCHAR2,
    point          IN SDO_GEOMETRY
    ) RETURN NUMBER;
```

Description

Returns the face ID number of the face that contains the specified point.

Parameters

topology
Name of the topology that contains the face and the point, or a null value, as explained in Section 2.1.3. Must not exceed 20 characters.

topo_map
Name of the TopoMap object, or a null value, as explained in Section 2.1.3. (TopoMap objects are explained in Section 2.1.1.)

point
Geometry object specifying the point.

Usage Notes

The `topology` or `topo_map` parameter should specify a valid name, as explained in Section 2.1.3.

This function determines, from the faces in the specified TopoMap object (including any island faces), which one face (if any) contains the specified point in its open set. (The open set of a face consists of all points inside, but not on the boundary of, the face.) If the point is exactly on the boundary of a face, the function returns a value of 0 (zero).

If the entire topology has been loaded into the TopoMap object and if the point is not in any finite face in the cache, this function returns a value of -1 (for the universal face). If a window from the topology has been loaded into the TopoMap object and if the point is not in any finite face in the cache, this function returns a value of -1 (for the universal face) if the point is inside the window and a value of 0 (zero) if the point is outside the window.

This function is equivalent to using the `getContainingFace` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the face ID number of the face that contains the point at (22, 37) in the `CITY_DATA_TOPOMAP` TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_CONTAINING_FACE(null, 'CITY_DATA_TOPOMAP', SDO_
GEOMETRY(2001, NULL, SDO_POINT_TYPE(22,37,NULL), NULL, NULL)) FROM DUAL;

SDO_TOPO_MAP.GET_CONTAINING_FACE(NULL,'CITY_DATA_TOPOMAP',SDO_
GEOMETRY(2001,NULL,SDO
--------------------------------------------------------------------------------
                                                                               2
```

SDO_TOPO_MAP.GET_EDGE_ADDITIONS

Format

SDO_TOPO_MAP.GET_EDGE_ADDITIONS() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of edge ID numbers of edges that have been added to the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the edge ID numbers of edges in the current updatable TopoMap object that have been added since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no additions during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getEdgeAdditions` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the edge ID numbers of edges that have been added to the current updatable TopoMap object.

```
SELECT SDO_TOPO_MAP.GET_EDGE_ADDITIONS FROM DUAL;

GET_EDGE_ADDITIONS
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(28, 29, 30, 32)
```

SDO_TOPO_MAP.GET_EDGE_CHANGES

Format

SDO_TOPO_MAP.GET_EDGE_CHANGES() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of edge ID numbers of edges that have been changed (modified) in the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the edge ID numbers of edges in the current updatable TopoMap object that have been changed since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no changes during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getEdgeChanges` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the edge ID numbers of edges that have been changed in the current updatable TopoMap object.

```
SELECT SDO_TOPO_MAP.GET_EDGE_CHANGES FROM DUAL;

GET_EDGE_CHANGES
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(3, 2, 1)
```

SDO_TOPO_MAP.GET_EDGE_COORDS

Format

```
SDO_TOPO_MAP.GET_EDGE_COORDS(
    topology    IN VARCHAR2,
    topo_map  IN VARCHAR2,
    edge_id     IN NUMBER
    ) RETURN SDO_NUMBER_ARRAY;
```

Description

Returns an array with the coordinates of the start node, shape points, and end node for the specified edge.

Parameters

topology
Name of the topology that contains the edge, or a null value, as explained in Section 2.1.3. Must not exceed 20 characters.

topo_map
Name of the TopoMap object, or a null value, as explained in Section 2.1.3. (TopoMap objects are explained in Section 2.1.1.)

edge_id
Edge ID value of the edge.

Usage Notes

The `topology` or `topo_map` parameter should specify a valid name, as explained in Section 2.1.3.

This function is equivalent to using the `getEdgeCoords` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the coordinates of the start node, shape points, and end node for the edge whose edge ID value is 1. The returned array contains coordinates for six points. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_EDGE_COORDS(null, 'CITY_DATA_TOPOMAP', 1) FROM DUAL;

SDO_TOPO_MAP.GET_EDGE_COORDS(NULL,'CITY_DATA_TOPOMAP',1)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(8, 30, 16, 30, 16, 38, 3, 38, 3, 30, 8, 30)
```

SDO_TOPO_MAP.GET_EDGE_DELETIONS

Format

SDO_TOPO_MAP.GET_EDGE_DELETIONS() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of edge ID numbers of edges that have been deleted from the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the edge ID numbers of edges in the current updatable TopoMap object that have been deleted since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no deletions during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getEdgeDeletions` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the edge ID numbers of edges that have been deleted from the current updatable TopoMap object. In this case, the return of an empty SDO_NUMBER_ARRAY object indicates that no edges have been deleted.

```
SELECT SDO_TOPO_MAP.GET_EDGE_DELETIONS FROM DUAL;

GET_EDGE_DELETIONS
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY()
```

SDO_TOPO_MAP.GET_EDGE_NODES

Format

```
SDO_TOPO_MAP.GET_EDGE_NODES(
    topology    IN VARCHAR2,
    topo_map  IN VARCHAR2,
    edge_id     IN NUMBER
    ) RETURN SDO_NUMBER_ARRAY;
```

Description

Returns an array with the ID numbers of the start and end nodes on the specified edge.

Parameters

topology
Name of the topology that contains the edge, or a null value, as explained in Section 2.1.3. Must not exceed 20 characters.

topo_map
Name of the TopoMap object, or a null value, as explained in Section 2.1.3. (TopoMap objects are explained in Section 2.1.1.)

edge_id
Edge ID value of the edge.

Usage Notes

The `topology` or `topo_map` parameter should specify a valid name, as explained in Section 2.1.3.

If the edge starts and ends at a node, the ID number of the node is the first and last number in the array.

This function has no exact equivalent method in the `TopoMap` class of the client-side Java API (described in Section 1.8.1). The `getEdge` method returns a Java edge object of the `oracle.spatial.topo.Edge` class.

Examples

The following example returns the ID numbers of the nodes on the edge whose edge ID value is 1. The returned array contains two nodes ID numbers, both of them 1, because the specified edge starts and ends at the node with node ID 1 and has a loop edge. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_EDGE_NODES(null, 'CITY_DATA_TOPOMAP', 1) FROM DUAL;

SDO_TOPO_MAP.GET_EDGE_NODES(NULL,'CITY_DATA_TOPOMAP',1)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(1, 1)
```

SDO_TOPO_MAP.GET_FACE_ADDITIONS

Format

SDO_TOPO_MAP.GET_FACE_ADDITIONS() RETURN SDO_NUMBER_ARRAY

Description

Returns an array of face ID numbers of faces that have been added to the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the face ID numbers of faces in the current updatable TopoMap object that have been added since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no additions during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getFaceAdditions` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the face ID numbers of faces that have been added to the current updatable TopoMap object.

```
SELECT SDO_TOPO_MAP.GET_FACE_ADDITIONS FROM DUAL;

GET_FACE_ADDITIONS
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(11)
```

SDO_TOPO_MAP.GET_FACE_CHANGES

Format

SDO_TOPO_MAP.GET_FACE_CHANGES() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of face ID numbers of faces that have been changed (modified) in the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the face ID numbers of faces in the current updatable TopoMap object that have been changed since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no changes during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getFaceChanges` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the face ID numbers of faces that have been changed in the current updatable TopoMap object.

```
SELECT SDO_TOPO_MAP.GET_FACE_CHANGES FROM DUAL;

GET_FACE_CHANGES
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(2, 1, -1)
```

SDO_TOPO_MAP.GET_FACE_DELETIONS

Format

SDO_TOPO_MAP.GET_FACE_DELETIONS() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of face ID numbers of faces that have been deleted from the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the face ID numbers of faces in the current updatable TopoMap object that have been deleted since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no deletions during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getFaceDeletions` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the face ID numbers of faces that have been deleted from the current updatable TopoMap object. In this case, the return of an empty SDO_NUMBER_ARRAY object indicates that no faces have been deleted.

```
SELECT SDO_TOPO_MAP.GET_FACE_DELETIONS FROM DUAL;

GET_FACE_DELETIONS
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY()
```

SDO_TOPO_MAP.GET_NEAREST_EDGE

Format

```
SDO_TOPO_MAP.GET_NEAREST_EDGE(
    topology    IN VARCHAR2,
    topo_map  IN VARCHAR2,
    point         IN SDO_GEOMETRY
    ) RETURN NUMBER;
```

Description

Returns the edge ID number of the edge that is nearest (closest to) the specified point.

Parameters

topology
Name of the topology that contains the edge and the point, or a null value, as explained in Section 2.1.3. Must not exceed 20 characters.

topo_map
Name of the TopoMap object, or a null value, as explained in Section 2.1.3. (TopoMap objects are explained in Section 2.1.1.)

point
Geometry object specifying the point.

Usage Notes

The `topology` or `topo_map` parameter should specify a valid name, as explained in Section 2.1.3.

The nearest edge is determined from the representation of the topology in the database, using the spatial index. If there are changed, added, or deleted edges in the instance and the database has not been updated to reflect those changes, the result may not reflect the true situation in the TopoMap object cache.

If multiple edges are equally close to the point, one of the edge ID values is returned.

This function is equivalent to using the getNearestEdge method of the TopoMap class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the edge ID number of the edge that is closest to the point at (8, 8) in the CITY_DATA_TOPOMAP TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_NEAREST_EDGE(null, 'CITY_DATA_TOPOMAP',
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(8,8,NULL), NULL, NULL))
  FROM DUAL;

SDO_TOPO_MAP.GET_NEAREST_EDGE(NULL,'CITY_DATA_TOPOMAP',SDO_GEOMETRY(2001,NULL,SD
--------------------------------------------------------------------------------
                                                                              22
```

SDO_TOPO_MAP.GET_NEAREST_NODE

Format

```
SDO_TOPO_MAP.GET_NEAREST_NODE(
    topology    IN VARCHAR2,
    topo_map  IN VARCHAR2,
    point        IN SDO_GEOMETRY
    ) RETURN NUMBER;
```

Description

Returns the node ID number of the node that is nearest (closest to) the specified point.

Parameters

topology
Name of the topology that contains the node and the point, or a null value, as explained in Section 2.1.3. Must not exceed 20 characters.

topo_map
Name of the TopoMap object, or a null value, as explained in Section 2.1.3. (TopoMap objects are explained in Section 2.1.1.)

point
Geometry object specifying the point.

Usage Notes

The `topology` or `topo_map` parameter should specify a valid name, as explained in Section 2.1.3.

The nearest node is determined from the representation of the topology in the database, using the spatial index. If there are changed, added, or deleted nodes in the instance and the database has not been updated to reflect those changes, the result may not reflect the true situation in the TopoMap object cache.

If multiple edges are equally close to the point, one of the edge ID values is returned.

This function is equivalent to using the getNearestNode method of the TopoMap class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the node ID number of the node that is closest to the point at (8, 8) in the CITY_DATA_TOPOMAP TopoMap object. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_NEAREST_NODE(null, 'CITY_DATA_TOPOMAP',
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(8,8,NULL), NULL, NULL))
  FROM DUAL;

SDO_TOPO_MAP.GET_NEAREST_NODE(NULL,'CITY_DATA_TOPOMAP',SDO_GEOMETRY(2001,NULL,SD
--------------------------------------------------------------------------------
                                                                               8
```

SDO_TOPO_MAP.GET_NODE_ADDITIONS

Format

SDO_TOPO_MAP.GET_NODE_ADDITIONS() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of node ID numbers of nodes that have been added to the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the node ID numbers of nodes in the current updatable TopoMap object that have been added since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no additions during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getNodeAdditions` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the node ID numbers of nodes that have been added to the current updatable TopoMap object.

```
SELECT SDO_TOPO_MAP.GET_NODE_ADDITIONS FROM DUAL;

GET_NODE_ADDITIONS
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(24, 25, 26, 27, 28)
```

SDO_TOPO_MAP.GET_NODE_CHANGES

Format

SDO_TOPO_MAP.GET_NODE_CHANGES() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of node ID numbers of nodes that have been changed (modified) in the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the node ID numbers of nodes in the current updatable TopoMap object that have been changed since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no changes during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getNodeChanges` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the node ID numbers of nodes that have been changed in the current updatable TopoMap object.

```
SELECT SDO_TOPO_MAP.GET_NODE_CHANGES FROM DUAL;

GET_NODE_CHANGES
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(2, 4)
```

SDO_TOPO_MAP.GET_NODE_COORD

Format

```
SDO_TOPO_MAP.GET_NODE_COORD(
     topology     IN VARCHAR2,
     topo_map  IN VARCHAR2,
     node_id      IN NUMBER
     ) RETURN SDO_POINT_TYPE;
```

Description

Returns an SDO_POINT_TYPE object with the coordinates of the specified node.

Parameters

topology
Name of the topology that contains the node, or a null value, as explained in Section 2.1.3. Must not exceed 20 characters.

topo_map
Name of the TopoMap object, or a null value, as explained in Section 2.1.3. (TopoMap objects are explained in Section 2.1.1.)

node_id
Node ID value of the node.

Usage Notes

The `topology` or `topo_map` parameter should specify a valid name, as explained in Section 2.1.3.

This function is equivalent to using the `getNodeCoord` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the coordinates of the node whose node ID value is 14. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_NODE_COORD(null, 'CITY_DATA_TOPOMAP', 14) FROM DUAL;

SDO_TOPO_MAP.GET_NODE_COORD(NULL,'CITY_DATA_TOPOMAP',14)(X, Y, Z)
--------------------------------------------------------------------------------
SDO_POINT_TYPE(21, 14, NULL)
```

SDO_TOPO_MAP.GET_NODE_DELETIONS

Format

SDO_TOPO_MAP.GET_NODE_DELETIONS() RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of node ID numbers of nodes that have been deleted from the current updatable TopoMap object.

Parameters

None.

Usage Notes

This function returns the node ID numbers of nodes in the current updatable TopoMap object that have been deleted since the object was most recently loaded (using SDO_TOPO_MAP.LOAD_TOPO_MAP), updated (using SDO_TOPO_MAP.UPDATE_TOPO_MAP), cleared (using SDO_TOPO_MAP.CLEAR_TOPO_MAP), committed (using SDO_TOPO_MAP.COMMIT_TOPO_MAP), or rolled back (using SDO_TOPO_MAP.ROLLBACK_TOPO_MAP). If there have been no deletions during that time, the function returns an empty SDO_NUMBER_ARRAY object.

This function is equivalent to using the `getNodeDeletions` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the node ID numbers of nodes that have been deleted from the current updatable TopoMap object. In this case, the return of an empty SDO_NUMBER_ARRAY object indicates that no nodes have been deleted.

```
SELECT SDO_TOPO_MAP.GET_NODE_DELETIONS FROM DUAL;

GET_NODE_DELETIONS
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY()
```

SDO_TOPO_MAP.GET_NODE_STAR

Format

```
SDO_TOPO_MAP.GET_NODE_STAR(
     topology     IN VARCHAR2,
     topo_map  IN VARCHAR2,
     node_id      IN NUMBER
     ) RETURN SDO_NUMBER_ARRAY;
```

Description

Returns an SDO_NUMBER_ARRAY object with the edge ID numbers, in clockwise order, of the edges that are connected to the specified node.

Parameters

topology
Name of the topology that contains the node, or a null value, as explained in Section 2.1.3. Must not exceed 20 characters.

topo_map
Name of the TopoMap object, or a null value, as explained in Section 2.1.3. (TopoMap objects are explained in Section 2.1.1.)

node_id
Node ID value of the node.

Usage Notes

The **node star** of a node is the edges that are connected to the node.

The `topology` or `topo_map` parameter should specify a valid name, as explained in Section 2.1.3.

This function is equivalent to using the `getNodeStar` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the node star of the node whose node ID value is 14. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_NODE_STAR(null, 'CITY_DATA_TOPOMAP', 14) FROM DUAL;

SDO_TOPO_MAP.GET_NODE_STAR(NULL,'CITY_DATA_TOPOMAP',14)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(19, -10, -20, -9)
```

SDO_TOPO_MAP.GET_TOPO_NAME

Format

```
SDO_TOPO_MAP.GET_TOPO_NAME(
     topo_map  IN VARCHAR2
     ) RETURN VARCHAR2;
```

Description

Returns the name of the topology associated with the specified TopoMap object.

Parameters

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

Usage Notes

This function is equivalent to using the `getTopoName` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example returns the name of the topology associated with the TopoMap object named `CITY_DATA_TOPOMAP`. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.GET_TOPO_NAME('CITY_DATA_TOPOMAP') FROM DUAL;

SDO_TOPO_MAP.GET_TOPO_NAME('CITY_DATA_TOPOMAP')
--------------------------------------------------------------------------------
CITY_DATA
```

SDO_TOPO_MAP.LIST_TOPO_MAPS

Format

SDO_TOPO_MAP.LIST_TOPO_MAPS() RETURN VARCHAR2;

Description

Returns a comma-delimited list of entries for each TopoMap object currently active in the session, or an empty string if there are no currently active TopoMap objects.

Parameters

None.

Usage Notes

Each entry in the comma-delimited list contains the following information: the name of the TopoMap object, the name of the topology associated with the TopoMap object, and either `updatable` if the TopoMap object can be updated (that is, edited) or `read-only` if the TopoMap object cannot be updated.

For more information about TopoMap objects, including updatable and read-only status, see Section 2.1.1.

To remove a TopoMap object from the session, use the SDO_TOPO_MAP.DROP_TOPO_MAP procedure.

Examples

The following example lists the Topomap object name, topology name, and whether the object is updatable or read-only for each TopoMap object currently active in the session. (The example refers to definitions and data from Section 1.11.)

```
SELECT SDO_TOPO_MAP.LIST_TOPO_MAPS FROM DUAL;

LIST_TOPO_MAPS
--------------------------------------------------------------------------------
(CITY_DATA_TOPOMAP, CITY_DATA, updatable)
```

SDO_TOPO_MAP.LOAD_TOPO_MAP

Format

```
SDO_TOPO_MAP.LOAD_TOPO_MAP(
    topo_map       IN VARCHAR2,
    allow_updates  IN VARCHAR2,
    build_indexes  IN VARCHAR2 DEFAULT 'TRUE'
    ) RETURN VARCHAR2;
```

or

```
SDO_TOPO_MAP.LOAD_TOPO_MAP(
    topo_map       IN VARCHAR2,
    xmin           IN NUMBER,
    ymin           IN NUMBER,
    xmax           IN NUMBER,
    ymax           IN NUMBER,
    allow_updates  IN VARCHAR2,
    build_indexes  IN VARCHAR2 DEFAULT 'TRUE'
    ) RETURN VARCHAR2;
```

Description

Loads an entire topology or a window (rectangular portion) of a topology into a TopoMap object; returns the string `TRUE` if topology objects were loaded into the cache, and `FALSE` if no topology objects were loaded into the cache.

Parameters

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

xmin
Lower-left X coordinate value for the window (rectangular portion of the topology) to be loaded.

See the Usage Notes and Figure 4–1 for information about which topology objects are loaded when you specify a window.

ymin
Lower-left Y coordinate value for the window (rectangular portion of the topology) to be loaded.

xmax
Upper-right X coordinate value for the window (rectangular portion of the topology) to be loaded.

ymax
Upper-right Y coordinate value for the window (rectangular portion of the topology) to be loaded.

allow_updates
`TRUE` makes the TopoMap object updatable; that is, it allows topology editing operations to be performed on the TopoMap object. `FALSE` makes the TopoMap object read-only; that is, it does not allow topology editing operations to be performed on the TopoMap object.

There can be no more than one updatable TopoMap object active in a user session.

build_indexes
`TRUE` (the default) builds in-memory R-tree indexes for edge and face data; `FALSE` does not build in-memory R-tree indexes for edge and face data. The indexes improve the performance of editing operations, especially with large topologies.

Usage Notes

You must create the TopoMap object (using the SDO_TOPO_MAP.CREATE_TOPO_MAP procedure) before you load data into it.

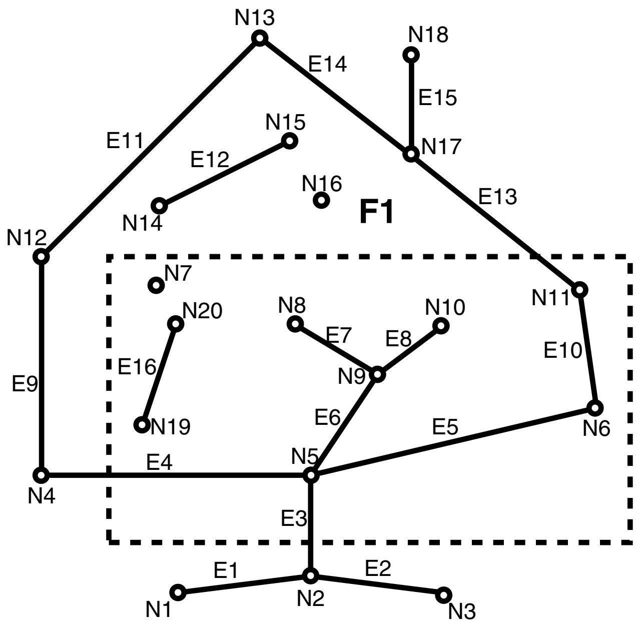
You cannot use this function if the TopoMap object already contains data. If the TopoMap object contains any data, you must do one of the following before calling this function: commit the changes (using the SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure) and clear the cache (using the SDO_TOPO_MAP.CLEAR_TOPO_MAP procedure), or roll back the changes (using the SDO_TOPO_MAP.ROLLBACK_TOPO_MAP procedure).

For information about using an in-memory cache to edit topology objects, see Section 2.1.

This function is equivalent to using the `loadTopoMap` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

If you specify a window using the `xmin`, `ymin`, `xmax`, and `ymax` parameters, all topology objects that are inside or on the window, all faces and edges that overlap the window, and all edges whose definitions depend on a face inside or partially within the window are loaded. For example, if an isolated node is on a face that is in the window, or if an edge has a left or right face that is in the window, it is loaded. Consider the topology and the window (shown by a dashed line) in Figure 4–1.

Figure 4–1 Loading Topology Objects into a Window



With the window shown in Figure 4–1:

- Face F1 is loaded because it is partially in the window.
- The following edges are loaded: E3, E4, E5, E6, E7, E8, E9, E10, E11, E12, E13, E14, E16.

Edges E1, E2, and E15 are not loaded, because their definitions do not refer to any face within the window (here, face F1).

- The following nodes are loaded: N5, N6, N7, N8, N9, N10, N11, N16, N19, N20.

  Non-isolated nodes N1, N2, N3, N4, N12, N13, N14, N15, N17, and N18 are not loaded, because they are not inside or on the window boundary.

Examples

The following example loads all `CITY_DATA` topology elements into its associated TopoMap object for editing (not read-only) and builds the in-memory R-tree indexes by default. It returns a result indicating that the operation was successful and that some topology objects were loaded into the cache. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.LOAD_TOPO_MAP('CITY_DATA_TOPOMAP', 'TRUE') INTO :res_varchar;

Call completed.

PRINT res_varchar;

RES_VARCHAR
--------------------------------------------------------------------------------
TRUE
```

SDO_TOPO_MAP.MOVE_EDGE

Format

```
SDO_TOPO_MAP.MOVE_EDGE(
     topology            IN VARCHAR2,
     edge_id             IN NUMBER,
     s_node_id           IN NUMBER,
     t_node_id           IN NUMBER,
     edge_coords         IN SDO_NUMBER_ARRAY);
```

or

```
SDO_TOPO_MAP.MOVE_EDGE(
     topology            IN VARCHAR2,
     edge_id             IN NUMBER,
     s_node_id           IN NUMBER,
     t_node_id           IN NUMBER,
     edge_coords         IN SDO_NUMBER_ARRAY,
     moved_iso_nodes  OUT SDO_NUMBER_ARRAY,
     moved_iso_edges  OUT SDO_NUMBER_ARRAY,
     allow_iso_moves   IN VARCHAR2);
```

Description

Moves a non-isolated edge.

Parameters

topology
Name of the topology in which to move the edge, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

edge_id
Edge ID of the edge to be moved.

edge_coords
An array of coordinates of the resulting moved edge, from start point to end point.

s_node_id
Node ID of the source node, which identifies the point (start node or end node of the edge) affected by the move, before the move occurs. For example, if the end point of edge E19 is to be moved from node N17 to node N16, the `s_node_id` value is the node ID number for node N17.

t_node_id
Node ID of the target node, which identifies the point affected by the move, after the move occurs. For example, if the end point of edge E19 is to be moved from node N17 to node N16, the `t_node_id` value is the node ID number for node N16.

moved_iso_nodes
Output parameter in which, if the `allow_iso_moves` parameter value is `TRUE`, Spatial stores the node ID values of any isolated nodes that have moved to a different face as a result of this procedure. If the `allow_iso_moves` parameter value is `FALSE`, Spatial stores the node ID values of any isolated nodes that did not move but that would have moved to a different face if the `allow_iso_moves` parameter value had been `TRUE`.

moved_iso_edges
Output parameter in which, if the `allow_iso_moves` parameter value is `TRUE`, Spatial stores the edge ID values of any isolated edges that have moved to a different face as a result of this procedure. If the `allow_iso_moves` parameter value is `FALSE`, Spatial stores the edge ID values of any isolated edges that did not move but that would have moved to a different face if the `allow_iso_moves` parameter value had been `TRUE`.

allow_iso_moves
`TRUE` causes Spatial to allow an edge move operation that would cause any isolated nodes or edges to be in a different face, and to adjust the containing face information for such isolated nodes and edges; `FALSE` causes Spatial not to allow an edge move operation that would cause any isolated nodes or edges to be in a different face.

If you use the format that does not include the `allow_iso_moves` parameter, Spatial allows an edge move operation that would cause any isolated nodes or edges to be in a different face, and it adjusts the containing face information for such isolated nodes and edges.

Usage Notes

For information about moving edges, see Section 2.3.2.

This function is equivalent to using the `moveEdge` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example moves the edge with edge ID value 19, and it displays the edge coordinates before and after the move. The edge move operation moves the end point of the edge from the node with node ID value 17 to the node with node ID value 16. (The edge being moved is E19 in Figure 1–2 in Section 1.2; and the edge is being changed from going vertically up to node N17, to going diagonally up and left to node N16. The example refers to definitions and data from Section 1.11.)

```
-- Get coordinates of edge E19.
SELECT SDO_TOPO_MAP.GET_EDGE_COORDS(null, 'CITY_DATA_TOPOMAP', 19) FROM DUAL;

SDO_TOPO_MAP.GET_EDGE_COORDS(NULL,'CITY_DATA_TOPOMAP',19)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(21, 14, 21, 22)

-- Move edge E19: from N14 -> N17 to N14 -> N16. The 3rd and 4th parameters
-- identify N17 and N16.
CALL SDO_TOPO_MAP.MOVE_EDGE(null, 19, 17, 16,
  SDO_NUMBER_ARRAY(21,14, 9,22));

Call completed.

-- Get coordinates of edge E19 after the move.
SELECT SDO_TOPO_MAP.GET_EDGE_COORDS(null, 'CITY_DATA_TOPOMAP', 19) FROM DUAL;

SDO_TOPO_MAP.GET_EDGE_COORDS(NULL,'CITY_DATA_TOPOMAP',19)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(21, 14, 9, 22)
```

SDO_TOPO_MAP.MOVE_ISOLATED_NODE

Format

```
SDO_TOPO_MAP.MOVE_ISOLATED_NODE(
    topology    IN VARCHAR2,
    node_id     IN NUMBER,
    point       IN SDO_GEOMETRY;
```

Description

Moves an isolated (island) node.

Parameters

topology
Name of the topology in which to move the node, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

node_id
Node ID of the node to be moved.

point
SDO_GEOMETRY object (point geometry) representing the location to which the isolated node is to be moved.

Usage Notes

For information about moving nodes, see Section 2.2.2.

This procedure is equivalent to using the `moveIsolatedNode` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example adds an isolated node and then moves it. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.ADD_ISOLATED_NODE(null, 2,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(22,38,NULL), NULL, NULL))
  INTO :res_number;
```

```
-- Move the just-added isolated node (from 20,38 to 22,39).
CALL SDO_TOPO_MAP.MOVE_ISOLATED_NODE( null, :res_number,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(22,39,NULL), NULL, NULL));
```

SDO_TOPO_MAP.MOVE_NODE

Format

```
SDO_TOPO_MAP.MOVE_NODE(
    topology        IN VARCHAR2,
    node_id         IN NUMBER,
    edges_coords    IN SDO_EDGE_ARRAY;
```

or

```
SDO_TOPO_MAP.MOVE_NODE(
    topology        IN VARCHAR2,
    node_id         IN NUMBER,
    edges_coords    IN SDO_EDGE_ARRAY,
    moved_iso_nodes OUT SDO_NUMBER_ARRAY,
    moved_iso_edges OUT SDO_NUMBER_ARRAY,
    allow_iso_moves IN VARCHAR2);
```

Description

Moves a non-isolated node and its attached edges.

Parameters

topology
Name of the topology in which to move the node, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

node_id
Node ID of the node to be moved.

edges_coords
An array of arrays, of type SDO_EDGE_ARRAY (described in Section 1.6.5). Each inner array consists of coordinates of each resulting attached edge, from start point to end point. The outer array consists of the attached edge arrays, starting with the start edge of the node to be moved and proceeding in clockwise order (with the

sequence of the edges as would be obtained in a call to the SDO_TOPO_MAP.GET_NODE_STAR function).

The array for each edge must include the start and end points. Any loops that connect twice at the moved node must be specified twice in the array.

moved_iso_nodes
Output parameter in which, if the `allow_iso_moves` parameter value is `TRUE`, Spatial stores the node ID values of any isolated nodes that have moved to a different face as a result of this procedure. If the `allow_iso_moves` parameter value is `FALSE`, Spatial stores the node ID values of any isolated nodes that did not move but that would have moved to a different face if the `allow_iso_moves` parameter value had been `TRUE`.

moved_iso_edges
Output parameter in which, if the `allow_iso_moves` parameter value is `TRUE`, Spatial stores the edge ID values of any isolated edges that have moved to a different face as a result of this procedure. If the `allow_iso_moves` parameter value is `FALSE`, Spatial stores the edge ID values of any isolated edges that did not move but that would have moved to a different face if the `allow_iso_moves` parameter value had been `TRUE`.

allow_iso_moves
`TRUE` causes Spatial to allow a node move operation that would cause any isolated nodes or edges to be in a different face, and to adjust the containing face information for such isolated nodes and edges; `FALSE` causes Spatial not to allow a node move operation that would cause any isolated nodes or edges to be in a different face.

If you use the format that does not include the `allow_iso_moves` parameter, Spatial allows a node move operation that would cause any isolated nodes or edges to be in a different face, and it adjusts the containing face information for such isolated nodes and edges.

Usage Notes

For information about moving nodes, see Section 2.2.2.

This procedure is equivalent to using the `moveNode` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example moves node N3 and adjusts the coordinates of the only attached edge. (The example refers to definitions and data from Section 1.11.)

```
-- Move node N3 to right: from 25,35 to 26,35.
-- E3 is changed from 25,30 -> 25,35 to 25,30 -> 26,35.
CALL SDO_TOPO_MAP.MOVE_NODE(null, 3,
  SDO_EDGE_ARRAY(SDO_NUMBER_ARRAY(25,30, 26,35)));
```

SDO_TOPO_MAP.REMOVE_EDGE

Format

```
SDO_TOPO_MAP.REMOVE_EDGE(
    topology    IN VARCHAR2,
    edge_id     IN NUMBER);
```

Description

Removes an edge from a topology.

Parameters

topology
Name of the topology from which to remove the edge, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

edge_id
Edge ID of the edge to be removed.

Usage Notes

Spatial automatically updates the appropriate entries in the <topology-name>_EDGE$ and <topology-name>_FACE$ tables.

For information about removing an edge from a topology, see Section 2.3.3.

Examples

The following example removes the edge with edge ID value 99 from the current updatable TopoMap object.

```
CALL SDO_TOPO_MAP.REMOVE_EDGE(null, 99);
```

SDO_TOPO_MAP.REMOVE_NODE

Format

```
SDO_TOPO_MAP.REMOVE_NODE(
    topology    IN VARCHAR2,
    node_id     IN NUMBER);
```

Description

Removes a node from a topology.

Parameters

topology
Name of the topology from which to remove the node, or null if you are using an updatable TopoMap object (see Section 2.1.2). Must not exceed 20 characters.

node_id
Node ID of the node to be removed.

Usage Notes

Spatial automatically updates the appropriate entries in the <topology-name>_NODE$ and <topology-name>_EDGE$ tables, and in the <topology-name>_FACE$ table if necessary.

For information about removing a node from a topology, see Section 2.2.3.

Examples

The following example removes the node with node ID value 500 from the current updatable TopoMap object.

```
CALL SDO_TOPO_MAP.REMOVE_NODE(null, 500);
```

SDO_TOPO_MAP.ROLLBACK_TOPO_MAP

Format

SDO_TOPO_MAP.ROLLBACK_TOPO_MAP;

Description

Rolls back all changes to the database that were made using the current updatable TopoMap object, discards any changes in the object, clears the object's cache structure, and makes the object read-only.

Parameters

None.

Usage Notes

Use this procedure when you are finished with a batch of edits to a topology and you want to discard (that is, not commit) all changes to the database and in the cache. After the rollback operation completes, you cannot edit the TopoMap object. To make further edits to the topology, you can load the topology into the same TopoMap object for update (using the SDO_TOPO_MAP.LOAD_TOPO_MAP procedure), or you can create a new TopoMap object (using the SDO_TOPO_MAP.CREATE_TOPO_MAP procedure) and load the topology into that TopoMap object for update.

To commit all TopoMap object changes, use the SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure.

For information about using an in-memory cache to edit topology objects, see Section 2.1.

This procedure is equivalent to using the `rollbackDB` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example rolls back from the database all changes associated with the current updatable TopoMap object.

```
EXECUTE SDO_TOPO_MAP.ROLLBACK_TOPO_MAP;
```

SDO_TOPO_MAP.UPDATE_TOPO_MAP

Format

```
SDO_TOPO_MAP.UPDATE_TOPO_MAP;
```

Description

Updates the topology to reflect edits made to the current updatable TopoMap object.

Parameters

None.

Usage Notes

Use this procedure to update the topology periodically during an editing session, as explained in Section 2.1.4. The TopoMap object remains open for further editing operations. The updates are not actually committed to the database until you call the SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure.

This procedure performs a level-0 validation of the TopoMap object before it updates the topology. (See the explanation of the `level` parameter for the SDO_TOPO_MAP.VALIDATE_TOPO_MAP function.)

If you caused in-memory R-tree indexes to be created when you loaded the TopoMap object (by specifying or accepting the default value of `TRUE` for the `build_indexes` parameter with the SDO_TOPO_MAP.LOAD_TOPO_MAP function), you can rebuild these indexes by using the SDO_TOPO_MAP.CREATE_EDGE_INDEX and SDO_TOPO_MAP.CREATE_FACE_INDEX procedures. For best index performance, these indexes should be rebuilt periodically when you are editing a large number of topology objects.

Contrast this procedure with the SDO_TOPO_MAP.CLEAR_TOPO_MAP procedure, which clears the cache associated with a specified TopoMap object and makes the object read-only.

To commit all TopoMap object changes, use the SDO_TOPO_MAP.COMMIT_TOPO_MAP procedure.

For information about using an in-memory cache to edit topology objects, see Section 2.1.

This procedure is equivalent to using the `updateTopology` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example updates the topology associated with the current updatable TopoMap object to reflect changes made to that object.

```
EXECUTE SDO_TOPO_MAP.UPDATE_TOPO_MAP;
```

SDO_TOPO_MAP.VALIDATE_TOPO_MAP

Format

```
SDO_TOPO_MAP.VALIDATE_TOPO_MAP(
     topo_map  IN VARCHAR2,
     level     IN NUMBER DEFAULT 1
     ) RETURN VARCHAR2;
```

Description

Performs a first-order validation of a TopoMap object, and optionally (by default) checks the computational geometry also; returns the string `TRUE` if the structure of the topology objects in TopoMap object is consistent, and raises an exception if the structure of the topology objects in TopoMap object is not consistent.

Parameters

topo_map
Name of the TopoMap object. (TopoMap objects are explained in Section 2.1.1.)

level
A value of 0 checks for the following conditions as part of a first-order validation:

- All faces are closed, and none have infinite loops.
- All previous and next edge pointers are consistent.
- All edges meet at nodes.
- Each island node is associated with a face.
- All edges on a face boundary are associated with the face.

A value of 1 (the default) checks for all conditions associated with a value of 0, plus the following conditions related to computational geometry:

- Each island is inside the boundary of its associated face.
- No edge intersects itself or another edge.
- Start and end coordinates of edges match coordinates of nodes.
- Node stars are properly ordered geometrically.

Usage Notes

This function checks the consistency of all pointer relationships among edges, nodes, and faces. You can use this function to validate an updatable TopoMap object before you update the topology (using the SDO_TOPO_MAP.UPDATE_TOPO_MAP procedure) or to validate a read-only TopoMap object before issuing queries.

This function is equivalent to using the `validateCache` method of the `TopoMap` class of the client-side Java API (described in Section 1.8.1).

Examples

The following example validates the topology in the TopoMap object named `CITY_DATA_TOPOMAP`, and it returns a result indicating that the topology is valid. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.VALIDATE_TOPO_MAP('CITY_DATA_TOPOMAP') INTO :res_varchar;

Call completed.

PRINT res_varchar;

RES_VARCHAR
--------------------------------------------------------------------------------
TRUE
```

SDO_TOPO_MAP.VALIDATE_TOPOLOGY

Format

```
SDO_TOPO_MAP.VALIDATE_TOPOLOGY(
    topology  IN VARCHAR2,
    ) RETURN VARCHAR2;
```

or

```
SDO_TOPO_MAP.VALIDATE_TOPOLOGY(
    topology          IN VARCHAR2,
    prevent_updates   IN VARCHAR2,
    level             IN NUMBER DEFAULT 1
    ) RETURN VARCHAR2;
```

or

```
SDO_TOPO_MAP.VALIDATE_TOPOLOGY(
    topology          IN VARCHAR2,
    xmin              IN NUMBER,
    ymin              IN NUMBER,
    xmax              IN NUMBER,
    ymax              IN NUMBER,
    prevent_updates   IN VARCHAR2,
    level             IN NUMBER DEFAULT 1
    ) RETURN VARCHAR2;
```

Description

Loads an entire topology or a window (rectangular portion) of a topology into a TopoMap object; returns the string `TRUE` if the structure of the topology is consistent, and raises an exception if the structure of the topology is not consistent.

Parameters

topology
Name of the topology to be validated. Must not exceed 20 characters.

xmin
Lower-left X coordinate value for the window (rectangular portion of the topology) to be validated.

ymin
Lower-left Y coordinate value for the window (rectangular portion of the topology) to be validated.

xmax
Upper-right X coordinate value for the window (rectangular portion of the topology) to be validated.

ymax
Upper-right Y coordinate value for the window (rectangular portion of the topology) to be validated.

prevent_updates
`TRUE` prevents other users from updating the topology while the validation is being performed; `FALSE` allows other users to update the topology while the validation is being performed. If you specify `FALSE`, any topology changes made by other users while the validation is being performed will not be considered by this function and will not affect the result.

level
A value of 0 checks for the following conditions:

- All faces are closed, and none have infinite loops.
- All previous and next edge pointers are consistent.
- All edges meet at nodes.
- Each island node is associated with a face.
- All edges on a face boundary are associated with the face.

A value of 1 (the default) checks for all conditions associated with a value of 0, plus the following conditions related to computational geometry:

- Each island is inside the boundary of its associated face.

- No edge intersects itself or another edge.
- Start and end coordinates of edges match coordinates of nodes.
- Node stars are properly ordered geometrically.

Usage Notes

This function implicitly creates a TopoMap object, and removes the object after the validation is complete. (TopoMap objects are described in Section 2.1.1.)

Examples

The following example validates the topology named `CITY_DATA`, and it returns a result indicating that the topology is valid. (The example refers to definitions and data from Section 1.11.)

```
CALL SDO_TOPO_MAP.VALIDATE_TOPOLOGY('CITY_DATA') INTO :res_varchar;

Call completed.

PRINT res_varchar;

RES_VARCHAR
--------------------------------------------------------------------------------
TRUE
```

5

Topology Operators

This chapter describes operators that you can use with Spatial topology data. For the current release, the only topology operator implemented is SDO_ ANYINTERACT.

To use any topology operator, you must understand the following:

- The conceptual information about topology in Chapter 1
- The information about Spatial operators in *Oracle Spatial User's Guide and Reference*

Table 5–1 lists the topology operators.

Table 5–1 Topology Operators

Function	Description
SDO_ANYINTERACT	Checks if any geometries in a topology geometry layer have the ANYINTERACT topological relationship with a specified topology geometry layer.

The rest of this chapter provides reference information about this operator.

SDO_ANYINTERACT

Format

SDO_ANYINTERACT(tg1, tg2);

Description

Checks if any geometries in a topology geometry layer have the ANYINTERACT topological relationship with a specified topology geometry layer.

Keywords and Parameters

Value	Description
tg1	Specifies a topology geometry layer. The topology geometry column must be spatially indexed. Data type is SDO_TOPO_GEOMETRY.
tg2	Specifies either a topology geometry layer or a spatial geometry layer. (It cannot be a transient instance of a topology geometry layer specified using a bind variable or SDO_TOPO_GEOMETRY constructor.) Data type is SDO_TOPO_GEOMETRY or SDO_GEOMETRY.

Returns

The expression SDO_ANYINTERACT(tg1,tg2) = '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 *Oracle Spatial User's Guide and Reference*.

Examples

The following example finds all street geometries that have the ANYINTERACT relationship with the land parcel named P3. (The examples for SDO_ ANYINTERACT use the data from Example 1–8 in Section 1.11.)

```
SELECT c.feature_name FROM city_streets c, land_parcels l
  WHERE l.feature_name = 'P3' AND
```

```
    SDO_ANYINTERACT (c.feature, l.feature) = 'TRUE';

FEATURE_NAME
------------------------------
R1
```

The following example finds all land parcel geometries that have the ANYINTERACT relationship with the traffic sign named S1.

```
SELECT l.feature_name FROM land_parcels l, traffic_signs t
  WHERE t.feature_name = 'S1' AND
    SDO_ANYINTERACT (l.feature, t.feature) = 'TRUE';

FEATURE_NAME
------------------------------
P1
P2
```

The following example finds all street geometries that have the ANYINTERACT relationship with a query window.

```
SQL> SELECT c.feature_name FROM city_streets c WHERE
  2    SDO_ANYINTERACT(
  3   c.feature,
  4   SDO_GEOMETRY(2003, NULL, NULL,
  5     SDO_ELEM_INFO_ARRAY(1, 1003, 3),
  6     SDO_ORDINATE_ARRAY(5,5, 30,40)))
  7  = 'TRUE';

FEATURE_NAME
------------------------------
R1
R3
R4
```

Part II

Network Data Model

This document has two main parts:

- Part I provides conceptual, usage, and reference information about the topology data model of Oracle Spatial.
- Part II provides conceptual, usage, and reference information about the network data model of Oracle Spatial.

Much of the conceptual information in Part I also applies to the network data model. Therefore, if you develop network applications, you should be familiar with the main terms and concepts from both parts of this document.

Part II contains the following chapters:

6 Network Data Model Overview

This chapter explains the concepts and operations related to the Oracle Spatial network data model. It assumes that you are familiar with the following information:

- The main topology concepts explained in Chapter 1, especially those related to nodes and links
- The main Oracle Spatial concepts, data types, and operations, as documented in *Oracle Spatial User's Guide and Reference*

Although this chapter discusses some network-related terms as they relate to Oracle Spatial, it assumes that you are familiar with basic network data modeling concepts.

This chapter contains the following major sections:

- Section 6.1, "Introduction to Network Modeling"
- Section 6.2, "Main Steps in Using the Network Data Model"
- Section 6.3, "Network Data Model Concepts"
- Section 6.4, "Network Applications"
- Section 6.5, "Network Hierarchy"
- Section 6.6, "Network Data Model Tables"
- Section 6.7, "Network Data Model Metadata Views"
- Section 6.8, "Network Data Model Application Programming Interface"
- Section 6.9, "Network Examples (PL/SQL)"

6.1 Introduction to Network Modeling

In many applications, capabilities or objects are modeled as nodes and links in a network. The network model contains logical information such as connectivity relationships among nodes and links, directions of links, and costs of nodes and links. With logical network information, you can analyze a network and answer questions, many of them related to path computing and tracing. For example, for a biochemical pathway, you can find all possible reaction paths between two chemical compounds; or for a road network, you can find the following information:

- What is the shortest (distance) or fastest (travel time) path between two cities?
- What is the closest hotel to a specific airport, and how can I get there?

In additional to logical network information, spatial information such as node locations and link geometries can be associated with the logical network. This information can help you to model the logical information (such as the cost of a route, because its physical length can be directly computed from its spatial representation).

The generic data model and network analysis capability can model and analyze many kinds of network applications in addition to traditional geographical information systems (GIS). For example, in biochemistry, applications may need to model reaction pathway networks for living organisms; and in the pharmaceutical industry, applications that model the drug discovery process may need to model protein-protein interaction.

The network modeling capabilities of Spatial include schema objects and an application programming interface (API). The schema objects include metadata and network tables. The API includes a server-side PL/SQL API (the SDO_NET package) for creating, managing, and analyzing networks in the database, and a middle-tier (or client-side) Java API for network analysis.

6.2 Main Steps in Using the Network Data Model

This section summarizes the main steps for working with the network data model in Oracle Spatial. It refers to important concepts, structures, and operations that are described in detail in other sections.

There are two basic approaches to creating a network:

- Let Spatial perform most operations, using procedures with names in the form CREATE_<*network-type*>_NETWORK. (See Section 6.2.1.)

- Perform the operations yourself: create the necessary network tables and update the network metadata. (See Section 6.2.2.)

With each approach, you must insert the network data into the network tables. You can then use the network data model PL/SQL and Java application programming interfaces (APIs) to update the network and perform other operations. (The PL/SQL and Java APIs are described in Section 6.8.)

6.2.1 Letting Spatial Perform Most Operations

To create a network by letting Spatial perform most of the necessary operations, follow these steps:

1. Create the network using a procedure with a name in the form CREATE_<*network-type*>_NETWORK, where <*network-type*> reflects the type of network that you want to create:

   - SDO_NET.CREATE_SDO_NETWORK for a spatial network with non-LRS SDO_GEOMETRY objects
   - SDO_NET.CREATE_LRS_NETWORK for a spatial network with LRS SDO_GEOMETRY objects
   - SDO_NET.CREATE_TOPO_NETWORK for a spatial network with topology geometry (SDO_TOPO_GEOMETRY) objects
   - SDO_NET.CREATE_LOGICAL_NETWORK for a logical network

   Each of these procedures creates the necessary network data model tables (described in Section 6.6) and inserts a row with the appropriate network metadata information into the xxx_SDO_NETWORK_METADATA views (described in Section 6.7.1).

   Each procedure has two formats: one format creates all network data model tables using default names for the tables and certain columns, and other format lets you specify names for the tables and certain columns. The default names for the network data model tables are <*network-name*>_NODE$, <*network-name*>_LINK$, <*network-name*>_PATH$, and <*network-name*>_PLINK$. The default name for cost columns in the network data model tables is COST, and the default name for geometry columns is GEOMETRY.

2. Insert data into the node and link tables, and if necessary into the path and path-link tables. (The node, link, path, and path-link tables are described in Section 6.6.)

3. Validate the network, using the SDO_NET.VALIDATE_NETWORK procedure.

4. For a spatial (SDO or LRS) network, insert the appropriate information into the USER_SDO_GEOM_METADATA view, and create spatial indexes on the geometry columns.

6.2.2 Performing the Operations Yourself

To create a network by performing the necessary operations yourself, follow these steps:

1. Create the node table, using the SDO_NET.CREATE_NODE_TABLE procedure. (The node table is described in Section 6.6.1.)
2. Insert data into the node table.
3. Create the link table, using the SDO_NET.CREATE_LINK_TABLE procedure. (The link table is described in Section 6.6.2).
4. Insert data into the link table.
5. Optionally, create the path table, using the SDO_NET.CREATE_PATH_TABLE procedure. (The path table is described in Section 6.6.3).
6. If you created the path table, create the path-link table, using the SDO_NET.CREATE_PATH_LINK_TABLE procedure. (The path-link table is described in Section 6.6.4).
7. If you created the path table and if you want to create paths, insert data into the table.
8. If you inserted data into the path table, insert the appropriate rows into the path-link table.
9. Insert a row into the USER_SDO_NETWORK_METADATA view with information about the network. (The USER_SDO_NETWORK_METADATA view is described in Section 6.7.1.)
10. For a spatial (SDO or LRS) network, insert the appropriate information into the USER_SDO_GEOM_METADATA view, and create spatial indexes on the geometry columns.
11. Validate the network, using the SDO_NET.VALIDATE_NETWORK procedure.

You can change the sequence of some of these steps. For example, you can create both the node and link tables first, and then insert data into each one; and you can insert the row into the USER_SDO_NETWORK_METADATA view before you create the node and link tables.

6.3 Network Data Model Concepts

A network is a type of mathematical graph that captures relationships between objects using connectivity. The connectivity may or may not be based on spatial proximity. For example, if two towns are on opposite sides of a lake, the shortest path based on spatial proximity (a straight line across the middle of the lake) is not relevant if you want to drive from one town to the other. Instead, to find the shortest driving distance, you need connectivity information about roads and intersections and about the "cost" of individual links.

A network consists of a set of nodes and links. Each link (sometimes also called an edge or a segment) specifies two nodes. A network can be directed or undirected, although links and paths typically have direction.

The following are some key terms related to the network data model:

- A **node** represents an object of interest.
- A **link** represents a relationship between two nodes. A link may be **directed** (that is, have a direction) or **undirected** (that is, not have a direction).
- A **path** is an alternating sequence of nodes and links, beginning and ending with nodes, and usually with no nodes and links appearing more than once. (Repeating nodes and links within a path are permitted, but are rare in most network applications.)
- A **network** is a set of nodes and links. A network is **directed** if the links that is contains are directed, and a network is **undirected** if the links that it contains are undirected.
- A **logical network** contains connectivity information but no geometric information. This is the model used for network analysis. A logical network can be treated as a directed graph or undirected graph, depending on the application.
- A **spatial network** contains both connectivity information and geometric information. In a spatial network, the nodes and links are SDO_GEOMETRY geometry objects without LRS information (an **SDO network**) or with LRS information (an **LRS network**), or SDO_TOPO_GEOMETRY objects (a **topology geometry network**).

  In an LRS network, each node includes a geometry ID value and a measure value, and each link includes a geometry ID value and start and end measure values; and the geometry ID value in each case refers to an SDO_GEOMETRY object with LRS information. A spatial network can be directed or undirected, depending on the application.

- A **feature** is an object of interest in a network application that is associated with a node or link. For example, in a transportation network, features include exits and intersections (mapped to nodes), and highways and streets (mapped to links).
- **Cost** is a non-negative numeric attribute that can be associated with links or nodes for computing the **minimum cost path**, which is the path that has the minimum total cost from a start node to an end node. You can specify a single cost factor, such as driving time or driving distance for links, in the network metadata.
- **Reachable nodes** are all nodes that can be reached from a given node. **Reaching nodes** are all nodes that can reach a given node.
- The **degree** of a node is the number of links to (that is, incident upon) the node. The **in-degree** is the number of inbound links, and the **out-degree** is the number of outbound links.
- **Network constraints** are restrictions defined on network analysis computations (for example, that driving routes must consist of expressways and major highways).
- A **spanning tree** of a connected graph is a tree (that is, a graph with no cycles) that connects all nodes of the graph. (The directions of links are ignored in a spanning tree.) The **minimum cost spanning tree** is the spanning tree that connects all nodes and has the minimum total cost.

6.4 Network Applications

Networks are used in applications to find how different objects are connected to each other. The connectivity is often expressed in terms of adjacency and path relationships. Two nodes are adjacent if they are connected by a link. There are often several paths between any two given nodes, and you may want to find the path with the minimum cost.

This section describes some typical examples of different kinds of network applications.

6.4.1 Road Network Example

In a typical road network, the intersections of roads are nodes and the road segments between two intersections are links. The spatial representation of a road is not inherently related to the nodes and links in the network. For example, a shape point in the spatial representation of a road (reflecting a sharp turn in the road) is not a node in the network if that shape point is not associated with an intersection; and a single spatial object may make up several links in a network (such as a straight segment intersected by three crossing roads). An important operation with a road network is to find the path from a start point to an end point, minimizing either the travel time or distance. There may be additional constraints on the path computation, such as having the path go through a particular landmark or avoid a particular intersection.

6.4.2 Train (Subway) Network Example

The subway network of any major city is probably best modeled as a logical network, assuming that precise spatial representation of the stops and track lines is unimportant. In such a network, all stops on the system constitute the nodes of the network, and a link is the connection between two stops if a train travels directly between these two stops. Important operations with a train network include finding all stations that can be reached from a specified station, finding the number of stops between two specified stations, and finding the travel time between two stations.

6.4.3 Utility Network Example

Utility networks, such as power line or cable networks, must often be configured to minimize the cost. An important operation with a utility network is to determine the connections among nodes, using minimum cost spanning tree algorithms, to provide the required quality of service at the minimum cost. Another important operation is reachability analysis, so that, for example, if a station in a water network is shut down, you know which areas will be affected.

6.4.4 Biochemical Network Example

Biochemical processes can be modeled as biochemical networks to represent reactions and regulations in living organisms. For example, metabolic pathways are networks involved in enzymatic reactions, while regulatory pathways represent protein-protein interactions. In this example, a pathway is a network; genes, proteins, and chemical compounds are nodes; and reactions among nodes are links. Important operations for a biochemical network include computing paths and the degrees of nodes.

6.5 Network Hierarchy

Some network applications require representations at different levels of abstraction. For example, two major processes might be represented as nodes with a link between them at the highest level of abstraction, and each major process might have several subordinate processes that are represented as nodes and links at the next level down.

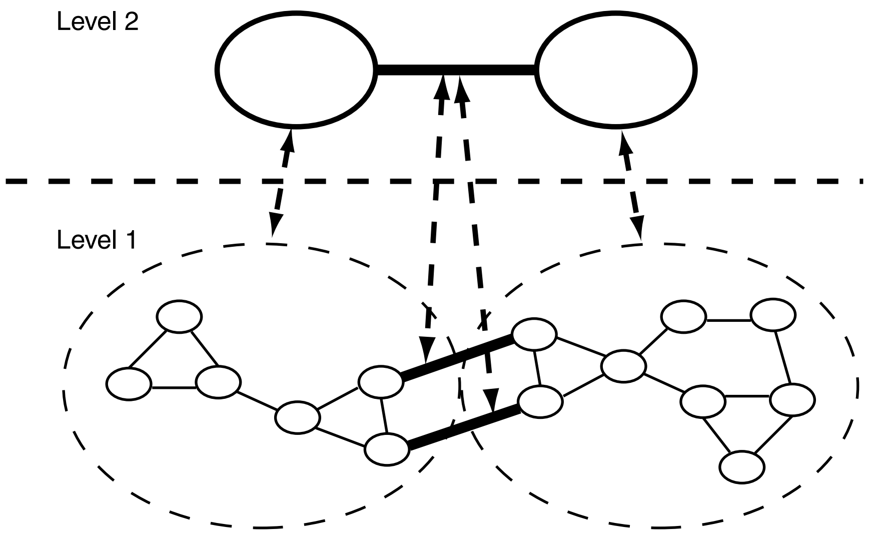
A **network hierarchy** allows you to represent a network with multiple levels of abstraction by assigning a hierarchy level to each node. (Links are not assigned a hierarchy level, and links can be between nodes in the same hierarchy level or in different levels.) The lowest (most detailed) level in the hierarchy is level 1, and successive higher levels are numbered 2, 3, and so on.

Nodes at adjacent levels of a network hierarchy have parent-child relationships. Each node at the higher level can be the parent node for one or more nodes at the lower level. Each node at the lower level can be a child node of one node at the higher level.

Links can also have parent-child relationships. However, because links are not assigned to a hierarchy level, there is not necessarily a relationship between link parent-child relationships and network hierarchy levels.

Figure 6–1 shows a simple hierarchical network, in which there are two levels.

Figure 6–1 Network Hierarchy



As shown in Figure 6–1:

- The top level (level 2) contains two nodes. Each node is the parent node of several nodes in the bottom level. The link between the nodes in the top level is the parent link of two links between nodes in the bottom level.
- The bottom level (level 1) shows the nodes that make up each node in the top level. It also shows the links between nodes that are child nodes of each parent node in the top level, and two links between nodes that have different parent nodes.
- The links between nodes in the bottom level that have different parent nodes are shown with dark connecting lines. These links are child links of the single link between the nodes in the top level in the hierarchy. (However, these two links in the bottom level could also be defined as not being child links of any parent link between nodes in a higher level.)
- The parent-child relationships between each parent node and link and its child nodes and links are shown with dashed lines with arrowheads at both ends.

Although it is not shown in Figure 6–1, links can cross hierarchy levels. For example, a link could be defined between a node in the top level and any node in the bottom level. In this case, there would not be a parent-child relationship between the links.

6.6 Network Data Model Tables

The connectivity information for a spatial network is stored in two tables: a node table and a link table. In addition, path information can be stored in a path table and a path-link table. You can have Spatial create these tables automatically when you create the network using a CREATE_<*network-type*>_NETWORK procedure; or you can create these tables using the SDO_NET.CREATE_NODE_TABLE, SDO_NET.CREATE_LINK_TABLE, SDO_NET.CREATE_PATH_TABLE, and SDO_NET.CREATE_PATH_LINK_TABLE procedures.

These tables contain columns with predefined names, and you must not change any of the predefined column names; however, you can add columns to the tables by using the ALTER TABLE statement with the ADD COLUMN clause. For example, although each link and path table is created with a single COST column, you can create additional columns and associate them with other comparable attributes. For example, if you wanted to assign a driving time, scenic appeal rating, and a danger rating to each link, you could use the COST column for driving time, add columns for SCENIC_APPEAL and DANGER to the link table, and populate all three columns with values to be interpreted by applications.

6.6.1 Node Table

Each network has a node table that can contain the columns described in Table 6–1. (The specific columns depend on the network type and whether the network is hierarchical or not.)

Table 6–1 Node Table Columns

Column Name	Data Type	Description
NODE_ID	NUMBER	ID number that uniquely identifies this node within the network.
NODE_NAME	VARCHAR2(32)	Name of the node.
NODE_TYPE	VARCHAR2(24)	User-defined string to identify the node type.
ACTIVE	VARCHAR2(1)	Contains `Y` if the node is active (visible in the network), or `N` if the node is not active.
PARTITION_ID	NUMBER	Reserved for future use.
<node_geometry_column>, or GEOM_ID and MEASURE	SDO_GEOMETRY, or SDO_TOPO_GEOMETRY, or NUMBER	For a spatial (SDO, non-LRS) network, name of the SDO_GEOMETRY column containing the geometry objects associated with the node. For a spatial topology network, name of the SDO_TOPO_GEOMETRY column containing the topology geometry objects associated with the node. For a spatial LRS network, GEOM_ID and MEASURE columns (both of type NUMBER) for the geometry objects associated with the node. For a logical network, this column is not used. For a spatial SDO or topology network, the actual column name is either a default name or what you specified as the `geom_column` parameter value in the call to the SDO_NET.CREATE_NODE_TABLE procedure.
<node_cost_column>	NUMBER	Name of the column containing the cost value to be associated with the node, for use by applications that use the network. The actual column name is either a default name or what you specified as the `cost_column` parameter value in the call to the SDO_NET.CREATE_NODE_TABLE procedure. The cost value can represent anything you want, for example, the toll to be paid at a toll booth.

Table 6–1 (Cont.) Node Table Columns

Column Name	Data Type	Description
HIERARCHY_LEVEL	NUMBER	For hierarchical networks only: number indicating the level in the network hierarchy for this node. (Section 6.5 explains network hierarchy.)
PARENT_NODE_ID	NUMBER	For hierarchical networks only: node ID of the parent node of this node. (Section 6.5 explains network hierarchy.)

6.6.2 Link Table

Each network has a link table that contains the columns described in Table 6–2.

Table 6–2 Link Table Columns

Column Name	Data Type	Description
LINK_ID	NUMBER	ID number that uniquely identifies this link within the network.
LINK_NAME	VARCHAR2(32)	Name of the link.
START_NODE_ID	NUMBER	Node ID of the node that starts the link.
END_NODE_ID	NUMBER	Node ID of the node that ends the link.
LINK_TYPE	VARCHAR2(24)	User-defined string to identify the link type.
ACTIVE	VARCHAR2(1)	Contains `Y` if the link is active (visible in the network), or `N` if the link is not active.
LINK_LEVEL	NUMBER	Priority level for the link; used for hierarchical modeling, so that links with higher priority levels can be considered first in computing a path.

Table 6–2 (Cont.) Link Table Columns

Column Name	Data Type	Description
<link_geometry_column>; or GEOM_ID, START_MEASURE, and END_MESURE	SDO_GEOMETRY, or SDO_TOPO_GEOMETRY, or NUMBER	For a spatial (SDO, non-LRS) network, name of the SDO_GEOMETRY column containing the geometry objects associated with the link. For a spatial topology network, name of the SDO_TOPO_GEOMETRY column containing the topology geometry objects associated with the link. For a spatial LRS network, GEOM_ID, START_MEASURE, and END_MEASURE columns (all of type NUMBER) for the geometry objects associated with the link. For a logical network, this column is not used. For a spatial SDO or topology network, the actual column name is either a default name or what you specified as the `geom_column` parameter value in the call to the SDO_NET.CREATE_LINK_TABLE procedure.
<link_cost_column>	NUMBER	Name of the column containing the cost value to be associated with the link, for use by applications that use the network. The actual column name is either a default name or what you specified as the `cost_column` parameter value in the call to the SDO_NET.CREATE_LINK_TABLE procedure. The cost value can represent anything you want, for example, the estimated driving time for the link.
PARENT_LINK_ID	NUMBER	For hierarchical networks only: link ID of the parent link of this link. (Section 6.5 explains parent-child relationships in a network hierarchy.)

6.6.3 Path Table

Each network can have a path table. A path is an ordered sequence of links, and is usually created as a result of network analysis. A path table provides a way to store the result of this analysis. For each path table, you must create an associated path-link table (described in Section 6.6.4). Each path table contains the columns described in Table 6–3.

Table 6–3 Path Table Columns

Column Name	Data Type	Description
PATH_ID	NUMBER	ID number that uniquely identifies this path within the network.
PATH_NAME	VARCHAR2(32)	Name of the path.
START_NODE_ID	NUMBER	Node ID of the node that starts the first link in the path.
END_NODE_ID	NUMBER	Node ID of the node that ends the last link in the path.
PATH_TYPE	VARCHAR2(24)	User-defined string to identify the path type.
COST	NUMBER	Cost value to be associated with the path, for use by applications that use the network. The cost value can represent anything you want, for example, the estimated driving time for the path.
SIMPLE	VARCHAR2(1)	Contains `Y` if the path is a simple path, or `N` if the path is a complex path. In a simple path, the links form an ordered list that can be traversed from the start node to the end node with each link visited once. In a complex path, there are multiple options for going from the start node to the end node.
<path_geometry_column>	SDO_GEOMETRY	For all network types except logical, name of the column containing the geometry object associated with the path. The actual column name is either a default name or what you specified as the `geom_column` parameter value in the call to the SDO_NET.CREATE_PATH_TABLE procedure. For a logical network, this column is not used.

6.6.4 Path-Link Table

For each path table (described in Section 6.6.3), you must create a path-link table. Each row in the path-link table uniquely identifies a link within a path in a network. The order of rows in the path-link table is not significant. Each path-link table contains the columns described in Table 6–4.

Table 6–4 Path-Link Table Columns

Column Name	Data Type	Description
PATH_ID	NUMBER	ID number of the path in the network.

Table 6–4 (Cont.) Path-Link Table Columns

Column Name	Data Type	Description
LINK_ID	NUMBER	ID number of the link in the network. Each combination of PATH_ID and LINK_ID must be unique within the network.
SEQ_NO	NUMBER	Unique sequence number of the link in the path. (The sequence numbers start at 1.) Sequence numbers allow paths to contain repeating nodes and links.

6.7 Network Data Model Metadata Views

There is a set of network metadata views for each schema (user): xxx_SDO_NETWORK_METADATA, where *xxx* can be USER or ALL. These views are created by Spatial.

6.7.1 xxx_SDO_NETWORK_METADATA Views

The following views contain information about networks:

- USER_SDO_NETWORK_METADATA contains information about all networks owned by the user.
- ALL_SDO_NETWORK_METADATA contains information about all networks on which the user has SELECT permission.

If you create a network using one of the CREATE_<*network-type*>_NETWORK procedures, the information in these views is automatically updated to reflect the new network; otherwise, you must insert information about the network into the USER_SDO_NETWORK_METADATA view.

The USER_SDO_NETWORK_METADATA and ALL_SDO_NETWORK_METADATA views contain the same columns, as shown Table 6–5, except that the USER_SDO_NETWORK_METADATA view does not contain the OWNER column. (The columns are listed in their order in the view definition.)

Table 6–5 Columns in the xxx_SDO_NETWORK_METADATA Views

Column Name	Data Type	Purpose
OWNER	VARCHAR2(32)	Owner of the network. (ALL_SDO_NETWORK_METADATA view only.)
NETWORK	VARCHAR2(32)	Name of the network.

Table 6–5 (Cont.) Columns in the xxx_SDO_NETWORK_METADATA Views

Column Name	Data Type	Purpose
NETWORK_CATEGORY	VARCHAR2(12)	Contains `SPATIAL` if the network nodes and links are associated with spatial geometries; contains `LOGICAL` if the network nodes and links are not associated with spatial geometries.
GEOMETRY_TYPE	VARCHAR2(20)	If `NETWORK_CATEGORY` is `SPATIAL`, contains a value indicating the geometry type of nodes and links: `SDO_GEOMETRY` for non-LRS SDO_GEOMETRY objects, `LRS_GEOMETRY` for LRS SDO_GEOMETRY objects, `TOPO_GEOMETRY` for SDO_TOPO_GEOMETRY objects.
NETWORK_TYPE	VARCHAR2(24)	User-defined string to identify the network type.
NO_OF_HIERARCHY_LEVELS	NUMBER	Number of levels in the network hierarchy. Contains 1 if there is no hierarchy. (See Section 6.5 for information about network hierarchy.)
NO_OF_PARTITIONS	NUMBER	(Must be 1 for the current release. Other values may be supported in future releases.)
LRS_TABLE_NAME	VARCHAR2(12)	If `GEOMETRY_TYPE` is `SDO_GEOMETRY`, contains the name of the table containing geometries associated with nodes.
LRS_GEOM_COLUMN	VARCHAR2(12)	If `LRS_TABLE_NAME` contains a table name, identifies the geometry column in that table.
NODE_TABLE_NAME	VARCHAR2(32)	If `GEOMETRY_TYPE` is `SDO_GEOMETRY`, contains the name of the table containing geometries associated with nodes. (The node table is described in Section 6.6.1.)
NODE_GEOM_COLUMN	VARCHAR2(32)	If `NODE_TABLE_NAME` contains a table name, identifies the geometry column in that table.
NODE_COST_COLUMN	VARCHAR2(32)	If `NODE_TABLE_NAME` contains a table name, identifies the cost column in that table.
LINK_TABLE_NAME	VARCHAR2(32)	If `GEOMETRY_TYPE` is `SDO_GEOMETRY`, contains the name of the table containing geometries associated with links. (The link table is described in Section 6.6.2.)
LINK_GEOM_COLUMN	VARCHAR2(32)	If `LINK_TABLE_NAME` contains a table name, identifies the geometry column in that table.

Table 6–5 (Cont.) Columns in the xxx_SDO_NETWORK_METADATA Views

Column Name	Data Type	Purpose
LINK_DIRECTION	VARCHAR2(12)	Contains a value indicating the type for all links in the network: `UNDIRECTED` or `DIRECTED`.
LINK_COST_ COLUMN	VARCHAR2(32)	If `LINK_TABLE_NAME` contains a table name, identifies the optional numeric column containing a cost value for each link.
PATH_TABLE_NAME	VARCHAR2(32)	Contains the name of an optional table containing information about paths. (The path table is described in Section 6.6.3.)
PATH_GEOM_ COLUMN	VARCHAR2(32)	If `PATH_TABLE_NAME` is associated with a spatial network, identifies the geometry column in that table.
PATH_LINK_TABLE_ NAME	VARCHAR2(32)	Contains the name of an optional table containing information about links for each path. (The path-link table is described in Section 6.6.4.)
PARTITION_TABLE_ NAME	VARCHAR2(32)	Reserved for future use.

6.8 Network Data Model Application Programming Interface

The Oracle Spatial network data model includes two client application programming interfaces (APIs): a PL/SQL interface provided by the SDO_NET package and a Java interface. Both interfaces let you create and update network data, and the Java interface provides network analysis capabilities. It is recommended that you use only PL/SQL or SQL to populate network tables and to create indexes, and that you use the Java interface for application development.

6.8.1 Network Data Model PL/SQL Interface

The SDO_NET package provides PL/SQL functions and procedures for creating, accessing, and managing networks on a database server. Example 6–3 in Section 6.9 shows the use of SDO_NET functions and procedures.

The SDO_NET functions and procedures can be grouped into the following logical categories:

- Creating networks:

  SDO_NET.CREATE_SDO_NETWORK

  SDO_NET.CREATE_LRS_NETWORK

SDO_NET.CREATE_TOPO_NETWORK

SDO_NET.CREATE_LOGICAL_NETWORK

- Copying and deleting networks:

  SDO_NET.COPY_NETWORK

  SDO_NET.DROP_NETWORK

- Creating network tables:

  SDO_NET.CREATE_NODE_TABLE

  SDO_NET.CREATE_LINK_TABLE

  SDO_NET.CREATE_PATH_TABLE

  SDO_NET.CREATE_PATH_LINK_TABLE

  SDO_NET.CREATE_LRS_TABLE

- Validating network objects:

  SDO_NET.VALIDATE_NETWORK

  SDO_NET.VALIDATE_NODE_SCHEMA

  SDO_NET.VALIDATE_LINK_SCHEMA

  SDO_NET.VALIDATE_PATH_SCHEMA

  SDO_NET.VALIDATE_LRS_SCHEMA

- Retrieving information (getting information about the network, checking for a characteristic):

  SDO_NET.GET_CHILD_LINKS

  SDO_NET.GET_CHILD_NODES

  SDO_NET.GET_GEOMETRY_TYPE

  SDO_NET.GET_IN_LINKS

  SDO_NET.GET_LINK_COST_COLUMN

  SDO_NET.GET_LINK_DIRECTION

  SDO_NET.GET_LINK_GEOM_COLUMN

  SDO_NET.GET_LINK_GEOMETRY

  SDO_NET.GET_LINK_TABLE_NAME

- SDO_NET.GET_LRS_GEOM_COLUMN
- SDO_NET.GET_LRS_LINK_GEOMETRY
- SDO_NET.GET_LRS_NODE_GEOMETRY
- SDO_NET.GET_LRS_TABLE_NAME
- SDO_NET.GET_NETWORK_TYPE
- SDO_NET.GET_NO_OF_HIERARCHY_LEVELS
- SDO_NET.GET_NO_OF_LINKS
- SDO_NET.GET_NO_OF_NODES
- SDO_NET.GET_NODE_DEGREE
- SDO_NET.GET_NODE_GEOM_COLUMN
- SDO_NET.GET_NODE_GEOMETRY
- SDO_NET.GET_NODE_IN_DEGREE
- SDO_NET.GET_NODE_OUT_DEGREE
- SDO_NET.GET_NODE_TABLE_NAME
- SDO_NET.GET_OUT_LINKS
- SDO_NET.GET_PATH_GEOM_COLUMN
- SDO_NET.GET_PATH_TABLE_NAME
- SDO_NET.IS_HIERARCHICAL
- SDO_NET.IS_LOGICAL
- SDO_NET.IS_SPATIAL
- SDO_NET.LRS_GEOMETRY_NETWORK
- SDO_NET.NETWORK_EXISTS
- SDO_NET.SDO_GEOMETRY_NETWORK
- SDO_NET.TOPO_GEOMETRY_NETWORK

For reference information about each SDO_NET function and procedure, see Chapter 7.

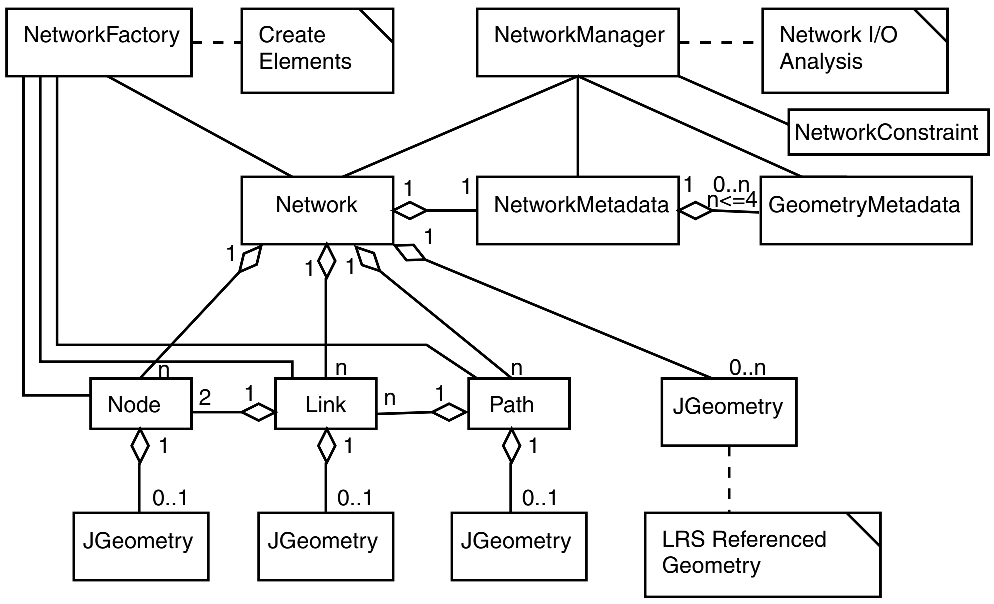
6.8.2 Network Data Model Java Interface

The Java client interface for the network data model consists of the following classes and interfaces:

- `NetworkManager`: class to load and store network data and metadata, and to perform network analysis
- `NetworkFactory`: class to create elements related to the network
- `NetworkConstraint`: class to create network constraints
- `Network`: interface for a network
- `NetworkMetadata`: interface for network metadata
- `GeometryMetadata`: class for geometry metadata
- `Node`: interface for a network node
- `Link`: interface for a network link
- `Path`: interface for a network path
- `MDPoint`: interface for a multiple-dimension point
- `MBR`: interface for a multiple-dimension minimum bounding rectangle
- `JGeometry`: class for Oracle Java SDO_GEOMETRY
- `NetworkDataException`: class for exceptions of network manager

Figure 6–2 is a Unified Modeling Language (UML) diagram that shows the relationship between the main classes and interfaces.

Figure 6–2 Java Classes and Interfaces for Network Data Model



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

6.8.2.1 Network Metadata and Data Management

You can use the Java API to perform network metadata and data management operations such as the following:

- Insert, delete, and modify node and link data
- Load a network from a database
- Store a network in a database
- Store network metadata in a database
- Modify network metadata attributes

6.8.2.2 Network Analysis

You can use the Java API to perform network analysis operations such as the following:

- Shortest path (for directed and undirected networks): typical transitive closure problems in graph theory. Given a start and an end node, find the shortest path.
- Minimum cost spanning tree (for undirected networks): Given an undirected graph, find the minimum cost tree that connects all nodes.
- Reachability: Given a node, find all nodes that can reach that node, or find all nodes that can be reached by that node.
- Within-cost analysis (for directed and undirected networks): Given a target node and a cost, find all nodes that can be reached by the target node within the given cost.
- Nearest-neighbors analysis (for directed and undirected networks): Given a target node and number of neighbors, find the neighbor nodes and their costs to go to the given target node.
- All paths between two nodes: Given two nodes, find all possible paths between them.
- "Traveling salesman problem" analysis: Given a set of nodes, find the lowest-cost path that visits all nodes and in which the start and end nodes are the same.

6.9 Network Examples (PL/SQL)

This section presents simplified examples that use SDO_NET functions and procedures. It includes the following sections:

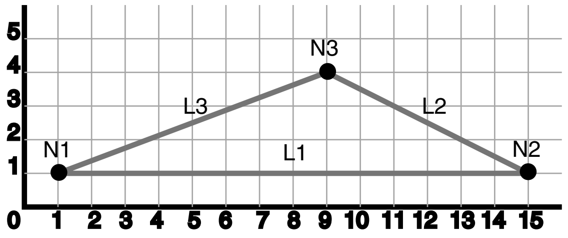
- Section 6.9.1, "Simple Spatial (SDO) Network Example"
- Section 6.9.2, "Simple Logical Network Example"
- Section 6.9.3, "Spatial (LRS) Network Example"
- Section 6.9.4, "Logical Hierarchical Network Example"

The examples refer to concepts that are explained in this chapter, and they use functions and procedures documented in Chapter 7.

6.9.1 Simple Spatial (SDO) Network Example

This section presents an example of a very simple spatial (SDO, not LRS) network that contains three nodes and a link between each node. The network is illustrated in Figure 6–3.

Figure 6–3 Simple Spatial (SDO) Network



As shown in Figure 6–3, node `N1` is at point 1,1, node `N2` is at point 15,1, and node `N3` is at point 9,4. Link `L1` is a straight line connecting nodes `N1` and `N2`, link `L2` is a straight line connecting nodes `N2` and `N3`, and link `L3` is a straight line connecting nodes `N3` and `N1`. There are no other nodes or shape points on any of the links.

Example 6–1 does the following:

- In a call to the SDO_NET.CREATE_SDO_NETWORK procedure, creates the `SDO_NET1` directed network; creates the SDO_NET1_NODE$, SDO_NET1_LINK$, SDO_NET1_PATH$, and SDO_NET1_PLINK$ tables; and updates the xxx_SDO_NETWORK_METADATA views. All geometry columns are named GEOMETRY. Both the node and link tables contain a cost column named COST.
- Populates the node, link, path, and path-link tables. It inserts three rows into the node table, three rows into the link table, two rows into the path table, and four rows into the path-link table.
- Updates the Oracle Spatial metadata, and creates spatial indexes on the GEOMETRY columns of the node and link tables. (These actions are not specifically related to network management, but that are necessary if applications are to benefit from spatial indexing on these geometry columns.)

Example 6–1 does not show the use of many SDO_NET functions and procedures; these are included in Example 6–3 in Section 6.9.3.

Example 6–1 Simple Spatial (SDO) Network Example (PL/SQL)

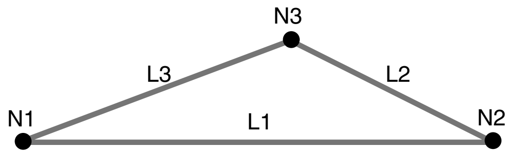
```
-- Create the SDO_NET1 directed network. Also creates the SDO_NET1_NODE$,
-- SDO_NET1_LINK$, SDO_NET1_PATH$, SDO_NET1_PLINK$ tables, and updates
-- USER_SDO_NETWORK_METADATA. All geometry columns are named GEOMETRY.
-- Both the node and link tables contain a cost column named COST.
EXECUTE SDO_NET.CREATE_SDO_NETWORK('SDO_NET1', 1, TRUE, TRUE);

-- Populate the SDO_NET1_NODE$ table.
-- N1
INSERT INTO sdo_net1_node$ VALUES(1, 'N1', NULL, 'Y', 1,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(1,1,NULL), NULL, NULL),
  5);
-- N2
INSERT INTO sdo_net1_node$ VALUES(2, 'N2', NULL, 'Y', 1,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(15,1,NULL), NULL, NULL),
  8);
-- N3
INSERT INTO sdo_net1_node$ VALUES(3, 'N3', NULL, 'Y', 1,
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(9,4,NULL), NULL, NULL),
  4);

-- Populate the SDO_NET1_LINK$ table.
-- L1
INSERT INTO sdo_net1_link$ VALUES(1, 'L1', 1, 2, NULL, 'Y', 1,
  SDO_GEOMETRY(2002, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1),
      SDO_ORDINATE_ARRAY(1,1, 15,1)),
  14);
-- L2
INSERT INTO sdo_net1_link$ VALUES(2, 'L2', 1, 3, NULL, 'Y', 1,
  SDO_GEOMETRY(2002, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1),
      SDO_ORDINATE_ARRAY(1,1, 9,4)),
  10);
-- L3
INSERT INTO sdo_net1_link$ VALUES(3, 'L3', 2, 3, NULL, 'Y', 1,
  SDO_GEOMETRY(2002, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1,2,1),
      SDO_ORDINATE_ARRAY(9, 4, 1,1)),
  10);

-- Do not populate the SDO_NET1_PATH$ and SDO_NET1_PLINK$ tables now.
-- Do this only when you need to create any paths.

---------------------------------------------------------------------------
-- REMAINING STEPS NEEDED TO USE SPATIAL INDEXES --
---------------------------------------------------------------------------
```

```
-- 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).
INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'SDO_NET1_NODE$',
  'GEOMETRY',
  SDO_DIM_ARRAY(   -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
     ),
  NULL   -- SRID (spatial reference system, also called coordinate system)
);
INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'SDO_NET1_LINK$',
  'GEOMETRY',
  SDO_DIM_ARRAY(   -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
     ),
  NULL   -- SRID (spatial reference system, also called coordinate system)
);

-- Create the spatial indexes
CREATE INDEX sdo_net1_nodes_idx ON sdo_net1_node$(geometry)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;
CREATE INDEX sdo_net1_links_idx ON sdo_net1_link$(geometry)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

6.9.2 Simple Logical Network Example

This section presents an example of a very simple logical network that contains three nodes and a link between the nodes. The network is illustrated in Figure 6–4.

Figure 6–4 Simple Logical Network



As shown in Figure 6–4, link L1 is a straight line connecting nodes N1 and N2, link L2 is a straight line connecting nodes N2 and N3, and link L3 is a straight line connecting nodes N3 and N1. There are no other nodes on any of the links.

Example 6–2 calls the SDO_NET.CREATE_LOGICAL_NETWORK procedure, which does the following: creates the LOG_NET1 directed network; creates the LOG_NET1_NODE$, LOG_NET1_LINK$, LOG_NET1_PATH$, and LOG_NET1_PLINK$ tables; and updates the xxx_SDO_NETWORK_METADATA views. Both the node and link tables contain a cost column named COST. (Because this is a logical network, there are no geometry columns.) The example also populates the node and link tables.

Example 6–2 does not show the use of many SDO_NET functions and procedures; these are included in the logical hierarchical network example (Example 6–4) in Section 6.9.4.

Example 6–2 Simple Logical Network Example (PL/SQL)

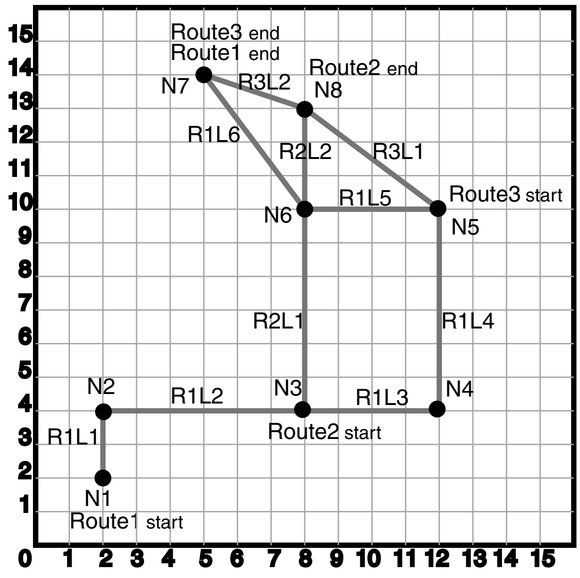
```
-- Create the LOG_NET1 directed logical network. Also creates the
-- LOG_NET1_NODE$, LOG_NET1_LINK$, LOG_NET1_PATH$,
-- and LOG_NET1_PLINK$ tables, and updates USER_SDO_NETWORK_METADATA.
-- Both the node and link tables contain a cost column named COST.
EXECUTE SDO_NET.CREATE_LOGICAL_NETWORK('LOG_NET1', 1, TRUE, TRUE);

-- Populate the LOG_NET1_NODE$ table.
-- N1
INSERT INTO log_net1_node$ (node_id, node_name, active, cost)
  VALUES (1, 'N1', 'Y', 2);
-- N2
INSERT INTO log_net1_node$ (node_id, node_name, active, cost)
  VALUES (2, 'N2', 'Y', 3);
-- N3
INSERT INTO log_net1_node$ (node_id, node_name, active, cost)
  VALUES (3, 'N3', 'Y', 2);

-- Populate the LOG_NET1_LINK$ table.
-- L1
INSERT INTO log_net1_link$ (link_id, link_name, start_node_id, end_node_id,
     active, link_level, cost)
  VALUES (1, 'L1', 1, 2, 'Y', 1, 10);
-- L2
INSERT INTO log_net1_link$ (link_id, link_name, start_node_id, end_node_id,
     active, link_level, cost)
  VALUES (2, 'L2', 2, 3, 'Y', 1, 7);
-- L3
```

```
INSERT INTO log_net1_link$ (link_id, link_name, start_node_id, end_node_id,
     active, link_level, cost)
  VALUES (3, 'L3', 3, 1, 'Y', 1, 8);

-- Do not populate the LOG_NET1_PATH$ and LOG_NET1_PLINK$ tables now.
-- Do this only when you need to create any paths.
```

6.9.3 Spatial (LRS) Network Example

This section presents an example of a spatial (LRS) network that uses the roads illustrated in Figure 6–5. This illustration is similar to the one used for the LRS example in *Oracle Spatial User's Guide and Reference*, but it adds two highways (`Route2` and `Route3`).

Figure 6–5 Roads for Spatial (LRS) Network Example



As shown in Figure 6–5:

- `Route1` starts at point 2,2 and ends at point 5,14. It has the following nodes: `N1`, `N2`, `N3`, `N4`, `N5`, `N6`, and `N7`. It has the following links: `R1L1`, `R1L2`, `R1L3`, `R1L4`, `R1L5`, and `R1L6`.

- `Route2` starts at point 8,4 and ends at point 8,13. It has the following nodes: `N3`, `N6`, and `N8`. It has the following links: `R2L1` and `R2L2`.
- `Route3` starts at point 12,10 and ends at point 5,14. It has the following nodes: `N5`, `N8`, and `N7`. It has the following links: `R3L1` and `R3L2`.

Example 6–3 does the following:

- Creates a table to hold the roads data.
- Inserts the definition of three roads into the table.
- Inserts the spatial metadata into the USER_SDO_GEOM_METADATA view.
- Creates a spatial index on the geometry column in the ROADS table.
- Creates and populates the node table.
- Creates and populates the link table.
- Creates and populates the path table and path-link table, for possible future use. (Before an application can use paths, you must populate these two tables.)
- Inserts network metadata into the USER_SDO_NETWORK_METADATA view.
- Uses various SDO_NET functions and procedures.

Example 6–3 Spatial (LRS) Network Example (PL/SQL)

```
---------------------------------------------------------------------------
-- CREATE AND POPULATE TABLE --
---------------------------------------------------------------------------
-- Create a table for roads. Use LRS.
CREATE TABLE roads (
  road_id  NUMBER PRIMARY KEY,
  road_name  VARCHAR2(32),
  road_geom  SDO_GEOMETRY,
  geom_id NUMBER);

-- Populate the table with roads (Route1, Route2, Route3).
INSERT INTO roads VALUES(
  1,
  'Route1',
  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(
      2,2,0,   -- Starting point - Node1; 0 is measure from start.
      2,4,2,   -- Node2; 2 is measure from start.
      8,4,8,   -- Node3; 8 is measure from start.
      12,4,12,  -- Node4; 12 is measure from start.
      12,10,NULL,  -- Node5; measure automatically calculated and filled.
      8,10,22,  -- Node6; 22 is measure from start.
      5,14,27)  -- Ending point - Node7; 27 is measure from start.
  ), 1001
);

INSERT INTO roads VALUES(
  2,
  'Route2',
  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(
      8,4,0,   -- Node3; 0 is measure from start.
      8,10,6,  -- Node6; 6 is measure from start.
      8,13,9)  -- Ending point - Node8; 9 is measure from start.
  ), 1002
);

INSERT INTO roads VALUES(
  3,
  'Route3',
  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(
      12,10,0,   -- Node5; 0 is measure from start.
      8,13,5,  -- Node8; 5 is measure from start.
      5,14,3.16)  -- Ending point - Node7; 8.16 is measure from start.
  ), 1003
);

---------------------------------------------------------------------------
-- UPDATE THE SPATIAL METADATA --
---------------------------------------------------------------------------
-- 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: roads and road_geom).
INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'ROADS',
  'ROAD_GEOM',
  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 (spatial reference system, also called coordinate system)
);

-------------------------------------------------------------------
-- CREATE THE SPATIAL INDEX --
-------------------------------------------------------------------
CREATE INDEX roads_idx ON roads(road_geom)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;

--------------------------------
-- USE SDO_NET SUBPROGRAMS
--------------------------------

-- This procedure does not use the CREATE_LRS_NETWORK procedure. Instead,
-- the user creates the network tables and populates the network metadata view.
-- Basic steps:
-- 1. Create and populate the node table.
-- 2. Create and populate the link table.
-- 3. Create the path table and path-link table (for possible
--    future use, before which they will need to be populated).
-- 4. Populate the network metadata (USER_SDO_NETWORK_METADATA).
--    Note: Can be done before or after Steps 1-3.
-- 5. Use various SDO_NET functions and procedures.

-- 1. Create and populate the node table.
EXECUTE SDO_NET.CREATE_NODE_TABLE('ROADS_NODES', 'LRS_GEOMETRY', 'NODE_
GEOMETRY', NULL, 1);

-- Populate the node table.

-- N1
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (1, 'N1', 'Y', 1001, 0);

-- N2
```

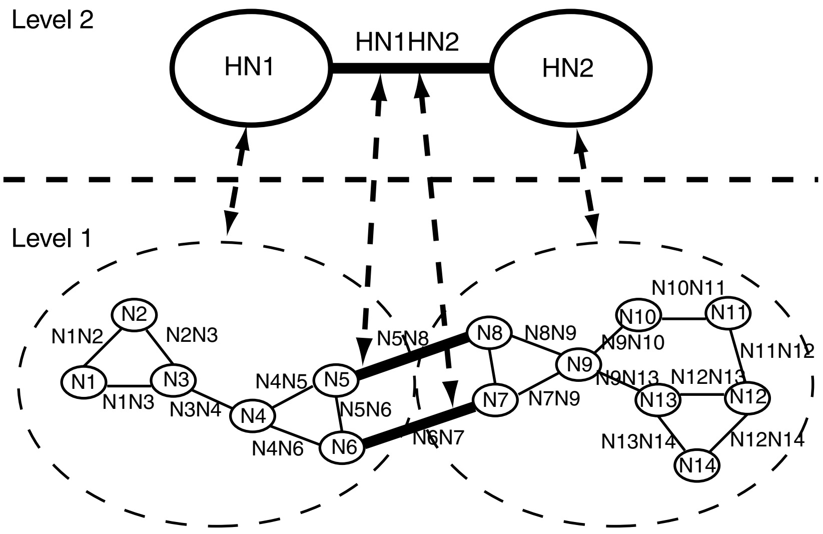
```
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (2, 'N2', 'Y', 1001, 2);

-- N3
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (3, 'N3', 'Y', 1001, 8);

-- N4
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (4, 'N4', 'Y', 1001, 12);

-- N5
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (5, 'N5', 'Y', 1001, 18);

-- N6
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (6, 'N6', 'Y', 1001, 22);

-- N7
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (7, 'N7', 'Y', 1001, 27);

-- N8
INSERT INTO roads_nodes (node_id, node_name, active, geom_id, measure)
  VALUES (8, 'N8', 'Y', 1002, 9);

-- 2. Create and populate the link table.
EXECUTE SDO_NET.CREATE_LINK_TABLE('ROADS_LINKS', 'LRS_GEOMETRY', 'LINK_
GEOMETRY', 'COST', 1);

-- Populate the link table.

-- Route1, Link1
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (101, 'R1L1', 1, 2, 'Y', 3, 1001, 0, 2);

-- Route1, Link2
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (102, 'R1L2', 2, 3, 'Y', 15, 1001, 2, 8);

 -- Route1, Link3
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (103, 'R1L3', 3, 4, 'Y', 10, 1001, 8, 12);

-- Route1, Link4
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (104, 'R1L4', 4, 5, 'Y', 15, 1001, 12, 18);

-- Route1, Link5
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (105, 'R1L5', 5, 6, 'Y', 10, 1001, 18, 22);

-- Route1, Link6
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (106, 'R1L6', 6, 7, 'Y', 7, 1001, 22, 27);

-- Route2, Link1 (cost = 30, a slow drive)
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (201, 'R2L1', 3, 6, 'Y', 30, 1002, 0, 6);

-- Route2, Link2
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (202, 'R2L2', 6, 8, 'Y', 5, 1002, 6, 9);

-- Route3, Link1
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (301, 'R3L1', 5, 8, 'Y', 5, 1003, 0, 5);

-- Route3, Link2
INSERT INTO roads_links (link_id, link_name, start_node_id, end_node_id, active,
   cost, geom_id, start_measure, end_measure)
VALUES (302, 'R3L2', 8, 7, 'Y', 5, 1003, 5, 8.16);

-- 3. Create the path table (to store created paths) and the path-link
--     table (to store links for each path) for possible future use,
--     before which they will need to be populated.
EXECUTE SDO_NET.CREATE_PATH_TABLE('ROADS_PATHS', 'PATH_GEOMETRY');
EXECUTE SDO_NET.CREATE_PATH_LINK_TABLE('ROADS_PATHS_LINKS');

-- 4. Populate the network metadata (USER_SDO_NETWORK_METADATA).
```

```
INSERT INTO user_sdo_network_metadata VALUES (
  'ROADS_NETWORK',  -- Network name
  'SPATIAL',  -- Network category
  'LRS_GEOMETRY',  -- Geometry type
  'Roadways',  -- Network type (user-defined)
  1,  -- No. of levels in hierarchy
  1,  -- No. of partitions
  'ROADS',   -- LRS table name
  'ROAD_GEOM' ,  -- LRS geometry column
  'ROADS_NODES',  -- Node table name
  'NODE_GEOMETRY',  -- Node geometry column
  'COST',  -- Node cost column
  'ROADS_LINKS',  -- Link table name
  'LINK_GEOMETRY',  -- Link geometry column
  'DIRECTED',  -- Link direction
  'COST',  -- Link cost column
  'ROADS_PATHS',  -- Path table name
  'PATH_GEOMETRY',  -- Path geometry column
  'ROADS_PATHS_LINKS',  -- Path-link table
  NULL  -- No partition table
  );

-- 5. Use various SDO_NET functions and procedures.

-- Validate the network.
SELECT SDO_NET.VALIDATE_NETWORK('ROADS_NETWORK') FROM DUAL;

-- Validate parts or aspects of the network.
SELECT SDO_NET.VALIDATE_LINK_SCHEMA('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.VALIDATE_LRS_SCHEMA('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.VALIDATE_NODE_SCHEMA('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.VALIDATE_PATH_SCHEMA('ROADS_NETWORK') FROM DUAL;

-- Retrieve various information (GET_xxx and some other functions).
SELECT SDO_NET.GET_CHILD_LINKS('ROADS_NETWORK', 101) FROM DUAL;
SELECT SDO_NET.GET_CHILD_NODES('ROADS_NETWORK', 1) FROM DUAL;
SELECT SDO_NET.GET_GEOMETRY_TYPE('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_IN_LINKS('ROADS_NETWORK', 3) FROM DUAL;
SELECT SDO_NET.GET_LINK_COST_COLUMN('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_LINK_DIRECTION('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_LINK_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_LINK_GEOMETRY('ROADS_NETWORK', 103) FROM DUAL;
SELECT SDO_NET.GET_LINK_TABLE_NAME('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_LRS_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_LRS_LINK_GEOMETRY('ROADS_NETWORK', 103) FROM DUAL;
SELECT SDO_NET.GET_LRS_NODE_GEOMETRY('ROADS_NETWORK', 3) FROM DUAL;
SELECT SDO_NET.GET_LRS_TABLE_NAME('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NETWORK_TYPE('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NO_OF_HIERARCHY_LEVELS('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NO_OF_LINKS('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NO_OF_NODES('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NO_OF_PARTITIONS('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NODE_DEGREE('ROADS_NETWORK', 3) FROM DUAL;
SELECT SDO_NET.GET_NODE_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NODE_GEOMETRY('ROADS_NETWORK', 3) FROM DUAL;
SELECT SDO_NET.GET_NODE_IN_DEGREE('ROADS_NETWORK', 3) FROM DUAL;
SELECT SDO_NET.GET_NODE_OUT_DEGREE('ROADS_NETWORK', 3) FROM DUAL;
SELECT SDO_NET.GET_NODE_TABLE_NAME('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_OUT_LINKS('ROADS_NETWORK', 3) FROM DUAL;
SELECT SDO_NET.GET_PARTITION_TABLE_NAME('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_PATH_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_PATH_TABLE_NAME('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.IS_HIERARCHICAL('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.IS_LOGICAL('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.IS_SPATIAL('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.LRS_GEOMETRY_NETWORK('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.NETWORK_EXISTS('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.SDO_GEOMETRY_NETWORK('ROADS_NETWORK') FROM DUAL;
SELECT SDO_NET.TOPO_GEOMETRY_NETWORK('ROADS_NETWORK') FROM DUAL;

-- Copy a network.
EXECUTE SDO_NET.COPY_NETWORK('ROADS_NETWORK', 'ROADS_NETWORK2');
```

6.9.4 Logical Hierarchical Network Example

This section presents an example of a logical network that contains the nodes and links illustrated in Figure 6–6. Because it is a logical network, there are no spatial geometries associated with it. (Figure 6–6 is essentially the same as Figure 6–1 in Section 6.5, but with the nodes and links labeled.)

Figure 6–6 Nodes and Links for Logical Network Example



As shown in Figure 6–6:

- The network is hierarchical, with two levels. The top level (level 2) consists of two nodes (`HN1` and `HN2`) and one link (`HN1HN2`) that links these nodes. The remaining nodes and links are in the bottom level (level 1) of the hierarchy.
- Each node in level 1 is a child node of one of the nodes in level 2. Node `HN1` has the following child nodes: `N1`, `N2`, `N3`, `N4`, `N5`, and `N6`. Node `HN2` has the following child nodes: `N7`, `N8`, `N9`, `N10`, `N11`, `N12`, `N13`, and `N14`.
- Two links (`N5N8` and `N6N7`) in level 1 are child links of the link `HN1HN2` in level 2.

Example 6–4 does the following:

- Creates and populates the node table.
- Creates and populates the link table.
- Creates and populates the path table and path-link table, for possible future use. (Before an application can use paths, you must populate these two tables.)
- Inserts network metadata into the USER_SDO_NETWORK_METADATA view.
- Uses various SDO_NET functions and procedures.

Example 6–4 Logical Network Example (PL/SQL)

```
-- Basic steps:
-- 1. Create and populate the node table.
-- 2. Create and populate the link table.
-- 3. Create the path table and path-link table (for possible
--    future use, before which they will need to be populated).
-- 4. Populate the network metadata (USER_SDO_NETWORK_METADATA).
--    Note: Can be done before or after Steps 1-3.
-- 5. Use various SDO_NET functions and procedures.

-- 1. Create and populate the node table.
EXECUTE SDO_NET.CREATE_NODE_TABLE('XYZ_NODES', NULL, NULL, NULL, 2);

-- Populate the node table, starting with the highest level in the hierarchy.

-- HN1 (Hierarchy level=2, highest in this network)
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level)
  VALUES (1, 'HN1', 'Y', 2);

-- HN2 (Hierarchy level=2, highest in this network)
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level)
  VALUES (2, 'HN2', 'Y', 2);

-- N1 (Hierarchy level 1, parent node ID = 1 for N1 through N6)
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (101, 'N1', 'Y', 1, 1);

-- N2
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (102, 'N2', 'Y', 1, 1);

-- N3
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (103, 'N3', 'Y', 1, 1);

-- N4
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (104, 'N4', 'Y', 1, 1);

-- N5
```

```
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (105, 'N5', 'Y', 1, 1);

-- N6
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (106, 'N6', 'Y', 1, 1);

-- N7 (Hierarchy level 1, parent node ID = 2 for N7 through N14)
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (107, 'N7', 'Y', 1, 2);

-- N8
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (108, 'N8', 'Y', 1, 2);

-- N9
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (109, 'N9', 'Y', 1, 2);

-- N10
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (110, 'N10', 'Y', 1, 2);

-- N11
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (111, 'N11', 'Y', 1, 2);

-- N12
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (112, 'N12', 'Y', 1, 2);

-- N13
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (113, 'N13', 'Y', 1, 2);

-- N14
```

```
INSERT INTO xyz_nodes (node_id, node_name, active, hierarchy_level,
     parent_node_id)
  VALUES (114, 'N14', 'Y', 1, 2);

-- 2. Create and populate the link table.
EXECUTE SDO_NET.CREATE_LINK_TABLE('XYZ_LINKS', NULL, NULL, 'COST', 2);

-- Populate the link table.

-- HN1HN2 (single link in highest hierarchy level: link level = 2)
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level)
  VALUES (1001, 'HN1HN2', 1, 2, 'Y', 2);

-- For remaining links, link level = 1 and cost (10, 20, or 30) varies among
links.
-- N1N2
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1101, 'N1N2', 101, 102, 'Y', 1, 10);

-- N1N3
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1102, 'N1N3', 101, 102, 'Y', 1, 20);

-- N2N3
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1103, 'N2N3', 102, 103, 'Y', 1, 30);

-- N3N4
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1104, 'N3N4', 103, 104, 'Y', 1, 10);

-- N4N5
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1105, 'N4N5', 104, 105, 'Y', 1, 20);

-- N4N6
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1106, 'N4N6', 104, 106, 'Y', 1, 30);
```

```
-- N5N6
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1107, 'N5N6', 105, 106, 'Y', 1, 10);

-- N5N8 (child of the higher-level link: parent ID = 1001)
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost, parent_link_id)
  VALUES (1108, 'N5N8', 105, 106, 'Y', 1, 20, 1001);

-- N6N7 (child of the higher-level link: parent ID = 1001)
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost, parent_link_id)
  VALUES (1109, 'N6N7', 106, 107, 'Y', 1, 30, 1001);

-- N7N9
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1110, 'N7N9', 107, 109, 'Y', 1, 10);

-- N8N9
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1111, 'N8N9', 108, 109, 'Y', 1, 20);

-- N9N10
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1112, 'N9N10', 109, 110, 'Y', 1, 30);

-- N9N13
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1113, 'N9N13', 109, 113, 'Y', 1, 10);

-- N10N11
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1114, 'N10N11', 110, 111, 'Y', 1, 20);

-- N11N12
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1115, 'N11N12', 111, 112, 'Y', 1, 30);
```

```
-- N12N13
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1116, 'N12N13', 112, 113, 'Y', 1, 10);

-- N12N14
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1117, 'N12N14', 112, 114, 'Y', 1, 20);

-- N13N14
INSERT INTO xyz_links (link_id, link_name, start_node_id, end_node_id, active,
     link_level, cost)
  VALUES (1118, 'N13N14', 113, 114, 'Y', 1, 30);

-- 3. Create the path table (to store created paths) and the path-link
--    table (to store links for each path) for possible future use,
--    before which they will need to be populated.
EXECUTE SDO_NET.CREATE_PATH_TABLE('XYZ_PATHS', NULL);
EXECUTE SDO_NET.CREATE_PATH_LINK_TABLE('XYZ_PATHS_LINKS');

-- 4. Populate the network metadata (USER_SDO_NETWORK_METADATA).

INSERT INTO user_sdo_network_metadata VALUES (
  'XYZ_NETWORK',  -- Network name
  'LOGICAL',   -- Network category
  NULL,  -- Null geom type because not a spatial network
  NULL, -- No user-specified network type string
  2,  -- No. of levels in hierarchy
  1,  -- No. of partitions
  NULL,   -- No LRS table name
  NULL,   -- No LRS geometry column
  'XYZ_NODES',  -- Node table name
  NULL,    -- No node geometry column
  NULL,  -- No node cost column
  'XYZ_LINKS',  -- Link table name
  NULL,   -- No link geometry column
  'DIRECTED',  -- Link direction
  'COST',  -- Link cost column
  'XYZ_PATHS',  -- Path table name
  NULL,  -- No path geometry column
  NULL,  -- No path-link table
  NULL  -- No partition table
  );
```

```
-- 5. Use various SDO_NET functions and procedures.

-- Validate the network.
SELECT SDO_NET.VALIDATE_NETWORK('XYZ_NETWORK') FROM DUAL;

-- Validate parts or aspects of the network.
SELECT SDO_NET.VALIDATE_LINK_SCHEMA('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.VALIDATE_LRS_SCHEMA('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.VALIDATE_NODE_SCHEMA('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.VALIDATE_PATH_SCHEMA('XYZ_NETWORK') FROM DUAL;

-- Retrieve various information (GET_xxx and some other functions).
SELECT SDO_NET.GET_CHILD_LINKS('XYZ_NETWORK', 1001) FROM DUAL;
SELECT SDO_NET.GET_CHILD_NODES('XYZ_NETWORK', 1) FROM DUAL;
SELECT SDO_NET.GET_CHILD_NODES('XYZ_NETWORK', 2) FROM DUAL;
SELECT SDO_NET.GET_IN_LINKS('XYZ_NETWORK', 104) FROM DUAL;
SELECT SDO_NET.GET_LINK_COST_COLUMN('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_LINK_DIRECTION('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_LINK_TABLE_NAME('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NETWORK_TYPE('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NO_OF_HIERARCHY_LEVELS('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NO_OF_LINKS('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NO_OF_NODES('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.GET_NODE_DEGREE('XYZ_NETWORK', 104) FROM DUAL;
SELECT SDO_NET.GET_NODE_IN_DEGREE('XYZ_NETWORK', 104) FROM DUAL;
SELECT SDO_NET.GET_NODE_OUT_DEGREE('XYZ_NETWORK', 104) FROM DUAL;
SELECT SDO_NET.GET_OUT_LINKS('XYZ_NETWORK', 104) FROM DUAL;
SELECT SDO_NET.GET_PATH_TABLE_NAME('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.IS_HIERARCHICAL('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.IS_LOGICAL('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.IS_SPATIAL('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.LRS_GEOMETRY_NETWORK('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.NETWORK_EXISTS('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.SDO_GEOMETRY_NETWORK('XYZ_NETWORK') FROM DUAL;
SELECT SDO_NET.TOPO_GEOMETRY_NETWORK('XYZ_NETWORK') FROM DUAL;

-- Copy a network.
EXECUTE SDO_NET.COPY_NETWORK('XYZ_NETWORK', 'XYZ_NETWORK2');
```

7 SDO_NET Package Subprograms

The MDSYS.SDO_NET package contains subprograms (functions and procedures) for managing networks. To use the subprograms in this chapter, you must understand the conceptual information in Chapter 6.

For a listing of the subprograms grouped in logical categories, see Section 6.8.1. The rest of this chapter provides reference information about the subprograms, listed in alphabetical order.

SDO_NET.COPY_NETWORK

Format

```
SDO_NET.COPY_NETWORK(
    source_network  IN VARCHAR2,
    target_network   IN VARCHAR2);
```

Description

Creates a copy of a network, including its metadata tables.

Parameters

source_network
Name of the network to be copied.

target_network
Name of the network to be created as a copy of `source_network`.

Usage Notes

This procedure creates an entry in the xxx_SDO_NETWORK_METADATA views (described in Section 6.7.1) for `target_network` that has the same information as for `source_network`, except for the new network name.

This procedure also creates a new node table, link table, and path table (if a path table exists for `source_network`) for `target_network` based on the metadata and data in these tables for `source_network`. These tables have names in the form <*target-network*>_NODE$, <*target-network*>_LINK$, and <*target-network*>_PATH$. For example, if `target_network` has the value `ROADS_NETWORK2` and if `source_network` has a path table, the names of the created metadata tables are `ROADS_NETWORK2_NODE$`, `ROADS_NETWORK2_LINK$`, and `ROADS_NETWORK2_PATH$`.

Examples

The following example creates a new network named `ROADS_NETWORK2` that is a copy of the network named `ROADS_NETWORK`.

```
EXECUTE SDO_NET.COPY_NETWORK('ROADS_NETWORK', 'ROADS_NETWORK2');
```

SDO_NET.CREATE_LINK_TABLE

Format

```
SDO_NET.CREATE_LINK_TABLE(
    table_name              IN VARCHAR2,
    geom_type               IN VARCHAR2,
    geom_column             IN VARCHAR2,
    cost_column             IN VARCHAR2,
    no_of_hierarchy_levels  IN NUMBER);
```

Description

Creates a link table for a network.

Parameters

table_name
Name of the link table.

geom_type
For a spatial network, specify a value indicating the geometry type of links: SDO_GEOMETRY for non-LRS SDO_GEOMETRY objects, LRS_GEOMETRY for LRS SDO_GEOMETRY objects, or TOPO_GEOMETRY for SDO_TOPO_GEOMETRY objects.

geom_column
For a spatial network, the name of the column containing the geometry objects associated with the links.

cost_column
Name of the column containing the cost values to be associated with the links.

no_of_hierarchy_levels
Number of hierarchy levels for links in the network. (For an explanation of network hierarchy, see Section 6.5.)

Usage Notes

The link table is described in Section 6.6.2.

Examples

The following example creates a link table named ROADS_LINKS, with a geometry column named LINK_GEOMETRY that will contain LRS geometries, a cost column named COST, and a single hierarchy level.

```
EXECUTE SDO_NET.CREATE_LINK_TABLE('ROADS_LINKS', 'LRS_GEOMETRY', 'LINK_
GEOMETRY', 'COST', 1);
```

SDO_NET.CREATE_LOGICAL_NETWORK

Format

```
SDO_NET.CREATE_LOGICAL_NETWORK(
    network                  IN VARCHAR2,
    no_of_hierarchy_levels  IN NUMBER,
    is_directed              IN BOOLEAN,
    node_with_cost           IN BOOLEAN DEFAULT FALSE);
```

or

```
SDO_NET.CREATE_LOGICAL_NETWORK(
    network                  IN VARCHAR2,
    no_of_hierarchy_levels  IN NUMBER,
    is_directed              IN BOOLEAN,
    node_table_name          IN VARCHAR2,
    node_cost_column         IN VARCHAR2,
    link_table_name          IN VARCHAR2,
    link_cost_column         IN VARCHAR2,
    path_table_name          IN VARCHAR2,
    path_link_table_name     IN VARCHAR2);
```

Description

Creates a logical network, creates all necessary tables, and updates the network metadata.

Parameters

network
Network name.

no_of_hierarchy_levels
Number of hierarchy levels for links in the network. (For an explanation of network hierarchy, see Section 6.5.)

is_directed
A Boolean value. `TRUE` indicates that the links are directed; `FALSE` indicates that the links are not directed.

node_with_cost
A Boolean value. `TRUE` causes a column named COST to be included in the <*network-name*>_NODE$ table; `FALSE` (the default) causes a column named COST not to be included in the <*network-name*>_NODE$ table.

node_table_name
Name of the node table to be created. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, a node table named <*network-name*>_NODE$ is created.

node_cost_column
Name of the cost column in the node table. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, the geometry column is named COST.

link_table_name
Name of the link table to be created. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, a link table named <*network-name*>_LINK$ is created.

link_cost_column
Name of the cost column in the link table. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, the geometry column is named COST.

path_table_name
Name of the path table to be created. (The path table is explained in Section 6.6.3.) If you use the format that does not specify this parameter, a path table named <*network-name*>_PATH$ is created.

path_link_table_name
Name of the path-link table to be created. (The path-link table is explained in Section 6.6.4.) If you use the format that does not specify this parameter, a path-link table named <*network-name*>_PLINK$ is created.

Usage Notes

This procedure provides a convenient way to create a logical network when the node, link, and optional related tables do not already exist. The procedure creates the network; creates the node, link, path, and path-link tables for the network; and inserts the appropriate information in the xxx_SDO_NETWORK_METADATA views (described in Section 6.7.1).

An exception is generated if any of the tables to be created already exists.

The procedure has two formats. The simpler format creates the tables using default values for the table name and the cost column name. The other format lets you specify names for the tables and the cost column.

As an alternative to using this procedure, you can create the network using the SDO_NET.CREATE_LOGICAL_NETWORK procedure; create the tables using the SDO_NET.CREATE_NODE_TABLE, SDO_NET.CREATE_LINK_TABLE, SDO_NET.CREATE_PATH_TABLE, and SDO_NET.CREATE_PATH_LINK_TABLE procedures; and insert the appropriate row in the USER_SDO_NETWORK_METADATA view.

Examples

The following example creates a directed logical network named `LOG_NET1`. The example creates the LOG_NET1_NODE$, LOG_NET1_LINK$,LOG_NET1_PATH$, and LOG_NET1_PLINK$ tables, and updates the xxx_SDO_NETWORK_METADATA views. Both the node and link tables contain a cost column named COST.

```
EXECUTE SDO_NET.CREATE_LOGICAL_NETWORK('LOG_NET1', 1, TRUE, TRUE);
```

SDO_NET.CREATE_LRS_NETWORK

Format

```
SDO_NET.CREATE_LRS_NETWORK(
    network                 IN VARCHAR2,
    lrs_table_name          IN VARCHAR2,
    lrs_geom_column         IN VARCHAR2,
    no_of_hierarchy_levels  IN NUMBER,
    is_directed             IN BOOLEAN,
    node_with_cost          IN BOOLEAN DEFAULT FALSE);
```

or

```
SDO_NET.CREATE_LRS_NETWORK(
    network                 IN VARCHAR2,
    no_of_hierarchy_levels  IN NUMBER,
    is_directed             IN BOOLEAN,
    node_table_name         IN VARCHAR2,
    node_cost_column        IN VARCHAR2,
    link_table_name         IN VARCHAR2,
    link_cost_column        IN VARCHAR2,
    lrs_table_name          IN VARCHAR2,
    lrs_geom_column         IN VARCHAR2,
    path_table_name         IN VARCHAR2,
    path_geom_column        IN VARCHAR2,
    path_link_table_name    IN VARCHAR2);
```

Description

Creates a spatial network containing LRS SDO_GEOMETRY objects, creates all necessary tables, and updates the network metadata.

Parameters

network
Network name.

lrs_table_name
Name of the table containing the LRS geometry column.

lrs_geom_column
Name of the column in `lrs_table_name` that contains LRS geometries (that is, SDO_GEOMETRY objects that include measure information for linear referencing).

is_directed
A Boolean value. `TRUE` indicates that the links are directed; `FALSE` indicates that the links are not directed.

no_of_hierarchy_levels
Number of hierarchy levels for links in the network. (For an explanation of network hierarchy, see Section 6.5.)

node_with_cost
A Boolean value. `TRUE` causes a column named COST to be included in the <*network-name*>_NODE$ table; `FALSE` (the default) causes a column named COST not to be included in the <*network-name*>_NODE$ table.

node_table_name
Name of the node table to be created. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, a node table named <*network-name*>_NODE$ is created.

node_cost_column
Name of the cost column in the node table. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, the geometry column is named COST.

link_table_name
Name of the link table to be created. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, a link table named <*network-name*>_LINK$ is created.

link_cost_column
Name of the cost column in the link table. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, the geometry column is named COST.

path_table_name
Name of the path table to be created. (The path table is explained in Section 6.6.3.) If you use the format that does not specify this parameter, a path table named <*network-name*>_PATH$ is created.

path_geom_column
Name of the geometry column in the path table. (The path table is explained in Section 6.6.3.) If you use the format that does not specify this parameter, the geometry column is named GEOMETRY.

path_link_table_name
Name of the path-link table to be created. (The path-link table is explained in Section 6.6.4.) If you use the format that does not specify this parameter, a path-link table named <*network-name*>_PLINK$ is created.

Usage Notes

This procedure provides a convenient way to create a spatial network of LRS geometries when the node, link, and optional related tables do not already exist. The procedure creates the network; creates the node, link, path, and path-link tables for the network; and inserts the appropriate information in the xxx_SDO_NETWORK_METADATA views (described in Section 6.7.1).

An exception is generated if any of the tables to be created already exists.

The procedure has two formats. The simpler format creates the tables using default values for the table name and the geometry and cost column names. The other format lets you specify names for the tables and the geometry and cost columns.

As an alternative to using this procedure, you can create the network using the SDO_NET.CREATE_LRS_NETWORK procedure; create the tables using the SDO_NET.CREATE_NODE_TABLE, SDO_NET.CREATE_LINK_TABLE, SDO_NET.CREATE_PATH_TABLE, and SDO_NET.CREATE_PATH_LINK_TABLE procedures; and insert the appropriate row in the USER_SDO_NETWORK_METADATA view.

Examples

The following example creates a directed spatial network named `LRS_NET1`. The LRS geometries are in the column named LRS_GEOM in the table named LRS_TAB. The example creates the LRS_NET1_NODE$, LRS_NET1_LINK$, LRS_NET1_PATH$, and LRS_NET1_PLINK$ tables, and updates the xxx_SDO_NETWORK_METADATA views. All geometry columns are named GEOMETRY. Both the node and link tables contain a cost column named COST.

```
EXECUTE SDO_NET.CREATE_LRS_NETWORK('LRS_NET1', 'LRS_TAB', 'LRS_GEOM', 1, TRUE,
TRUE);
```

SDO_NET.CREATE_LRS_TABLE

Format

SDO_NET.CREATE_LRS_TABLE(

table_name IN VARCHAR2,

geom_column IN VARCHAR2);

Description

Creates a table for storing Oracle Spatial linear referencing system (LRS) geometries.

Parameters

table_name
Name of the table containing the geometry column specified in `geom_column`.

geom_column
Name of the column (of type SDO_GEOMETRY) to contain geometry objects.

Usage Notes

This procedure creates a table named `table_name` with two columns: GEOM_ID of type NUMBER and `geom_column` of type SDO_GEOMETRY.

Although the created table does not need to be used to store LRS geometries, the procedure is intended as a convenient method for creating a table to store such geometries. You will probably want to modify the table to add other columns before you store data in the table.

Examples

The following example creates a table named HIGHWAYS with a geometry column named GEOM.

```
EXECUTE SDO_NET.CREATE_LRS_TABLE('HIGHWAYS', 'GEOM');

PL/SQL procedure successfully completed.

DESCRIBE highways
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
```

```
GEOM_ID                                   NOT NULL NUMBER
GEOM                                               MDSYS.SDO_GEOMETRY
```

SDO_NET.CREATE_NODE_TABLE

Format

```
SDO_NET.CREATE_NODE_TABLE(
     table_name              IN VARCHAR2,
     geom_type               IN VARCHAR2,
     geom_column             IN VARCHAR2,
     cost_column             IN VARCHAR2,
     no_of_hierarchy_levels  IN NUMBER);
```

Description

Creates a node table.

Parameters

table_name
Name of the node table.

geom_type
For a spatial network, specify a value indicating the geometry type of nodes: `SDO_GEOMETRY` for non-LRS SDO_GEOMETRY objects, `LRS_GEOMETRY` for LRS SDO_GEOMETRY objects, or `TOPO_GEOMETRY` for SDO_TOPO_GEOMETRY objects.

geom_column
For a spatial network, the name of the column containing the geometry objects associated with the nodes.

cost_column
Name of the column containing the cost values to be associated with the nodes.

no_of_hierarchy_levels
Number of hierarchy levels for nodes in the network. (For an explanation of network hierarchy, see Section 6.5.)

Usage Notes

The node table is described in Section 6.6.1.

Examples

The following example creates a node table named ROADS_NODES with a geometry column named NODE_GEOMETRY that will contain LRS geometries, no cost column, and a single hierarchy level.

```
EXECUTE SDO_NET.CREATE_NODE_TABLE('ROADS_NODES', 'LRS_GEOMETRY', 'NODE_
GEOMETRY', NULL, 1);
```

SDO_NET.CREATE_PATH_LINK_TABLE

Format

```
SDO_NET.CREATE_PATH_LINK_TABLE(
    table_name  IN VARCHAR2);
```

Description

Creates a path-link table, that is, a table with a row for each link in each path in the path table.

Parameters

table_name
Name of the path-link table.

Usage Notes

The path-link table is described in Section 6.6.4.

To use paths with a network, you must populate the path-link table.

Examples

The following example creates a path-link table named ROADS_PATHS_LINKS.

```
EXECUTE SDO_NET.CREATE_PATH_LINK_TABLE('ROADS_PATHS_LINKS');
```

SDO_NET.CREATE_PATH_TABLE

Format

```
SDO_NET.CREATE_PATH_TABLE(
    table_name  IN VARCHAR2,
    geom_column  IN VARCHAR2);
```

Description

Creates a path table.

Parameters

table_name
Name of the path table.

geom_column
For a spatial network, name of the column containing the geometry objects associated with the paths.

Usage Notes

The path table is described in Section 6.6.3.

To use paths with a network, after you create the path table, you must create the path-link table using the SDO_NET.CREATE_PATH_LINK_TABLE procedure, and populate the path-link table.

Examples

The following example creates a path table named ROADS_PATHS that contains a geometry column named PATH_GEOMETRY.

```
EXECUTE SDO_NET.CREATE_PATH_TABLE('ROADS_PATHS', 'PATH_GEOMETRY');
```

SDO_NET.CREATE_SDO_NETWORK

Format

```
SDO_NET.CREATE_SDO_NETWORK(
    network                 IN VARCHAR2,
    no_of_hierarchy_levels  IN NUMBER,
    is_directed             IN BOOLEAN,
    node_with_cost          IN BOOLEAN DEFAULT FALSE);
```

or

```
SDO_NET.CREATE_SDO_NETWORK(
    network                 IN VARCHAR2,
    no_of_hierarchy_levels  IN NUMBER,
    is_directed             IN BOOLEAN,
    node_table_name         IN VARCHAR2,
    node_geom_column        IN VARCHAR2,
    node_cost_column        IN VARCHAR2,
    link_table_name         IN VARCHAR2,
    link_geom_column        IN VARCHAR2,
    link_cost_column        IN VARCHAR2,
    path_table_name         IN VARCHAR2,
    path_geom_column        IN VARCHAR2,
    path_link_table_name    IN VARCHAR2);
```

Description

Creates a spatial network containing non-LRS SDO_GEOMETRY objects, creates all necessary tables, and updates the network metadata.

Parameters

network
Network name.

no_of_hierarchy_levels
Number of hierarchy levels for links in the network. (For an explanation of network hierarchy, see Section 6.5.)

is_directed
A Boolean value. `TRUE` indicates that the links are directed; `FALSE` indicates that the links are not directed.

node_with_cost
A Boolean value. `TRUE` causes a column named COST to be included in the <*network-name*>_NODE$ table; `FALSE` (the default) causes a column named COST not to be included in the <*network-name*>_NODE$ table.

node_table_name
Name of the node table to be created. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, a node table named <*network-name*>_NODE$ is created.

node_geom_column
Name of the geometry column in the node table. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, the geometry column is named GEOMETRY.

node_cost_column
Name of the cost column in the node table. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, the geometry column is named COST.

link_table_name
Name of the link table to be created. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, a link table named <*network-name*>_LINK$ is created.

link_geom_column
Name of the geometry column in the link table. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, the geometry column is named GEOMETRY.

link_cost_column
Name of the cost column in the link table. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, the geometry column is named COST.

path_table_name
Name of the path table to be created. (The path table is explained in Section 6.6.3.) If you use the format that does not specify this parameter, a path table named <*network-name*>_PATH$ is created.

path_geom_column
Name of the geometry column in the path table. (The path table is explained in Section 6.6.3.) If you use the format that does not specify this parameter, the geometry column is named GEOMETRY.

path_link_table_name
Name of the path-link table to be created. (The path-link table is explained in Section 6.6.4.) If you use the format that does not specify this parameter, a path-link table named <*network-name*>_PLINK$ is created.

Usage Notes

This procedure provides a convenient way to create a spatial network when the node, link, and optional related tables do not already exist. The procedure creates the network; creates the node, link, path, and path-link tables for the network; and inserts the appropriate information in the xxx_SDO_NETWORK_METADATA views (described in Section 6.7.1).

An exception is generated if any of the tables to be created already exists.

The procedure has two formats. The simpler format creates the tables using default values for the table name and the geometry and cost column names. The other format lets you specify names for the tables and the geometry and cost columns.

As an alternative to using this procedure, you can create the network using the SDO_NET.CREATE_SDO_NETWORK procedure; create the tables using the SDO_NET.CREATE_NODE_TABLE, SDO_NET.CREATE_LINK_TABLE, SDO_NET.CREATE_PATH_TABLE, and SDO_NET.CREATE_PATH_LINK_TABLE procedures; and insert the appropriate row in the USER_SDO_NETWORK_METADATA view.

Examples

The following example creates a directed spatial network named `SDO_NET1`. The example creates the SDO_NET1_NODE$, SDO_NET1_LINK$, SDO_NET1_PATH$, and SDO_NET1_PLINK$ tables, and updates the xxx_SDO_NETWORK_ METADATA views. All geometry columns are named GEOMETRY. Both the node and link tables contain a cost column named COST.

```
EXECUTE SDO_NET.CREATE_SDO_NETWORK('SDO_NET1', 1, TRUE, TRUE);
```

SDO_NET.CREATE_TOPO_NETWORK

Format

```
SDO_NET.CREATE_TOPO_NETWORK(
    network                  IN VARCHAR2,
    no_of_hierarchy_levels   IN NUMBER,
    is_directed              IN BOOLEAN,
    node_with_cost           IN BOOLEAN DEFAULT FALSE);
```

or

```
SDO_NET.CREATE_TOPO_NETWORK(
    network                  IN VARCHAR2,
    no_of_hierarchy_levels   IN NUMBER,
    is_directed              IN BOOLEAN,
    node_table_name          IN VARCHAR2,
    node_cost_column         IN VARCHAR2,
    link_table_name          IN VARCHAR2,
    link_cost_column         IN VARCHAR2,
    path_table_name          IN VARCHAR2,
    path_geom_column         IN VARCHAR2,
    path_link_table_name     IN VARCHAR2);
```

Description

Creates a spatial topology network containing SDO_TOPO_GEOMETRY objects, creates all necessary tables, and updates the network metadata.

Parameters

network
Network name.

no_of_hierarchy_levels
Number of hierarchy levels for links in the network. (For an explanation of network hierarchy, see Section 6.5.)

is_directed
A Boolean value. `TRUE` indicates that the links are directed; `FALSE` indicates that the links are not directed.

node_with_cost
A Boolean value. `TRUE` causes a column named COST to be included in the <*network-name*>_NODE$ table; `FALSE` (the default) causes a column named COST not to be included in the <*network-name*>_NODE$ table.

node_table_name
Name of the node table to be created. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, a node table named <*network-name*>_NODE$ is created.

node_cost_column
Name of the cost column in the node table. (The node table is explained in Section 6.6.1.) If you use the format that does not specify this parameter, the geometry column is named COST.

link_table_name
Name of the link table to be created. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, a link table named <*network-name*>_LINK$ is created.

link_cost_column
Name of the cost column in the link table. (The link table is explained in Section 6.6.2.) If you use the format that does not specify this parameter, the geometry column is named COST.

path_table_name
Name of the path table to be created. (The path table is explained in Section 6.6.3.) If you use the format that does not specify this parameter, a path table named <*network-name*>_PATH$ is created.

path_geom_column
Name of the geometry column in the path table. (The path table is explained in Section 6.6.3.) If you use the format that does not specify this parameter, the geometry column is named GEOMETRY.

path_link_table_name
Name of the path-link table to be created. (The path-link table is explained in Section 6.6.4.) If you use the format that does not specify this parameter, a path-link table named <*network-name*>_PLINK$ is created.

Usage Notes

This procedure provides a convenient way to create a spatial network when the node, link, and optional related tables do not already exist. The procedure creates the network; creates the node, link, path, and path-link tables for the network; and inserts the appropriate information in the xxx_SDO_NETWORK_METADATA views (described in Section 6.7.1). The node and link tables contain a topology geometry column named TOPO_GEOMETRY of type SDO_TOPO_GEOMETRY.

An exception is generated if any of the tables to be created already exists.

The procedure has two formats. The simpler format creates the tables using default values for the table name and the geometry and cost column names. The other format lets you specify names for the tables and the geometry and cost columns.

As an alternative to using this procedure, you can create the network using the SDO_NET.CREATE_TOPO_NETWORK procedure; create the tables using the SDO_NET.CREATE_NODE_TABLE, SDO_NET.CREATE_LINK_TABLE, SDO_NET.CREATE_PATH_TABLE, and SDO_NET.CREATE_PATH_LINK_TABLE procedures; and insert the appropriate row in the USER_SDO_NETWORK_METADATA view.

Examples

The following example creates a directed spatial topology geometry network named `TOPO_NET1`. The example creates the TOPO_NET1_NODE$, TOPO_NET1_LINK$, TOPO_NET1_PATH$, and TOPO_NET1_PLINK$ tables, and updates the xxx_SDO_NETWORK_METADATA views. The topology geometry columns are named TOPO_GEOMETRY. Both the node and link tables contain a cost column named COST.

```
EXECUTE SDO_NET.CREATE_TOPO_NETWORK('TOPO_NET1', 1, TRUE, TRUE);
```

SDO_NET.DROP_NETWORK

Format

```
SDO_NET.DROP_NETWORK(
    network  IN VARCHAR2);
```

Description

Drops a network.

Parameters

network
Name of the network to be dropped.

Usage Notes

This procedure also deletes the node, link, and path tables associated with the network, and the network metadata for the network.

Examples

The following example drops the network named `ROADS_NETWORK`.

```
EXECUTE SDO_NET.DROP_NETWORK('ROADS_NETWORK');
```

SDO_NET.GET_CHILD_LINKS

Format

SDO_NET.GET_CHILD_LINKS(

network IN VARCHAR2,

link_id IN NUMBER) RETURN SDO_NUMBER_ARRAY;

Description

Returns the child links of a link.

Parameters

network
Network name.

link_id
ID of the link for which to return the child links.

Usage Notes

For information about parent and child nodes and links in a network hierarchy, see Section 6.5.

Examples

The following example returns the child links of the link in the `XYZ_NETWORK` network whose link ID is 1001.

```
SELECT SDO_NET.GET_CHILD_LINKS('XYZ_NETWORK', 1001) FROM DUAL;

SDO_NET.GET_CHILD_LINKS('XYZ_NETWORK',1001)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(1108, 1109)
```

SDO_NET.GET_CHILD_NODES

Format

SDO_NET.GET_CHILD_NODES(

network IN VARCHAR2,

node_id IN NUMBER) RETURN SDO_NUMBER_ARRAY;

Description

Returns the child nodes of a node.

Parameters

network
Network name.

link_id
ID of the node for which to return the child nodes.

Usage Notes

For information about parent and child nodes and links in a network hierarchy, see Section 6.5.

Examples

The following example returns the child nodes of the node in the `XYZ_NETWORK` network whose node ID is 1.

```
SELECT SDO_NET.GET_CHILD_NODES('XYZ_NETWORK', 1) FROM DUAL;

SDO_NET.GET_CHILD_NODES('XYZ_NETWORK',1)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(101, 102, 103, 104, 105, 106)
```

SDO_NET.GET_GEOMETRY_TYPE

Format

SDO_NET.GET_GEOMETRY_TYPE(

network IN VARCHAR2) RETURN VARCHAR2;

Description

Returns the geometry type for a spatial network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the GEOMETRY_TYPE column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the geometry type for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_GEOMETRY_TYPE('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_GEOMETRY_TYPE('ROADS_NETWORK')
--------------------------------------------------------------------------------
LRS_GEOMETRY
```

SDO_NET.GET_IN_LINKS

Format

SDO_NET.GET_IN_LINKS(

network IN VARCHAR2,

node_id IN NUMBER) RETURN SDO_NUMBER_ARRAY ;

Description

Returns an array of link ID numbers of the inbound links to a node.

Parameters

network
Network name.

node_id
ID of the node for which to return the array of inbound links.

Usage Notes

For information about inbound links and related network data model concepts, see Section 6.3.

Examples

The following example returns an array of link ID numbers of the inbound links into the node whose node ID is 3 in the network named ROADS_NETWORK.

```
SELECT SDO_NET.GET_IN_LINKS('ROADS_NETWORK', 3) FROM DUAL;

SDO_NET.GET_IN_LINKS('ROADS_NETWORK',3)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(102)
```

SDO_NET.GET_LINK_COST_COLUMN

Format

```
SDO_NET.GET_LINK_COST_COLUMN(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns the name of the link cost column for a network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the LINK_COST_COLUMN column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the link cost column for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_LINK_COST_COLUMN('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_LINK_COST_COLUMN('ROADS_NETWORK')
--------------------------------------------------------------------------------
COST
```

SDO_NET.GET_LINK_DIRECTION

Format

```
SDO_NET.GET_LINK_DIRECTION(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns the link direction for a network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the LINK_DIRECTION column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the link direction for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_LINK_DIRECTION('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_LINK_DIRECTION('ROADS_NETWORK')
--------------------------------------------------------------------------------
DIRECTED
```

SDO_NET.GET_LINK_GEOM_COLUMN

Format

```
SDO_NET.GET_LINK_GEOM_COLUMN(
    network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns the name of the link geometry column for a spatial network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the LINK_GEOM_COLUMN column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the link geometry column for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_LINK_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_LINK_GEOM_COLUMN('ROADS_NETWORK')
--------------------------------------------------------------------------------
LINK_GEOMETRY
```

SDO_NET.GET_LINK_GEOMETRY

Format

```
SDO_NET.GET_LINK_GEOMETRY(
    network  IN VARCHAR2,
    link_id  IN NUMBER) RETURN MDSYS.SDO_GEOMETRY;
```

Description

Returns the geometry associated with a link in a spatial network.

Parameters

network
Network name.

link_id
ID number of the link for which to return the geometry.

Usage Notes

None.

Examples

The following example returns the geometry associated with the link whose link ID is 103 in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_LINK_GEOMETRY('ROADS_NETWORK', 103) FROM DUAL;

SDO_NET.GET_LINK_GEOMETRY('ROADS_NETWORK',103)(SDO_GTYPE, SDO_SRID, SDO_POINT(X,
--------------------------------------------------------------------------------
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
8, 4, 12, 4))
```

SDO_NET.GET_LINK_TABLE_NAME

Format

SDO_NET.GET_LINK_TABLE_NAME(

network IN VARCHAR2) RETURN VARCHAR2;

Description

Returns the name of the link table for a network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the LINK_TABLE_NAME column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the link table for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_LINK_TABLE_NAME('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_LINK_TABLE_NAME('ROADS_NETWORK')
--------------------------------------------------------------------------------
ROADS_LINKS
```

SDO_NET.GET_LRS_GEOM_COLUMN

Format

SDO_NET.GET_LRS_GEOM_COLUMN(

network IN VARCHAR2) RETURN VARCHAR2;

Description

Returns the name of the LRS geometry column for a spatial network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the LRS_GEOM_COLUMN column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the LRS geometry column for the network named ROADS_NETWORK.

```
SELECT SDO_NET.GET_LRS_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_LRS_GEOM_COLUMN('ROADS_NETWORK')
--------------------------------------------------------------------------------
ROAD_GEOM
```

SDO_NET.GET_LRS_LINK_GEOMETRY

Format

```
SDO_NET.GET_LRS_LINK_GEOMETRY(
    network  IN VARCHAR2,
    link_id  IN NUMBER) RETURN MDSYS.SDO_GEOMETRY;
```

Description

Returns the LRS geometry associated with a link in a spatial LRS network.

Parameters

network
Network name.

link_id
ID number of the link for which to return the geometry.

Usage Notes

None.

Examples

The following example returns the LRS geometry associated with the link whose link ID is 103 in the network named ROADS_NETWORK.

```
SELECT SDO_NET.GET_LRS_LINK_GEOMETRY('ROADS_NETWORK', 103) FROM DUAL;

SDO_NET.GET_LRS_LINK_GEOMETRY('ROADS_NETWORK',103)(SDO_GTYPE, SDO_SRID, SDO_POIN
--------------------------------------------------------------------------------
SDO_GEOMETRY(2002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
8, 4, 12, 4))
```

SDO_NET.GET_LRS_NODE_GEOMETRY

Format

```
SDO_NET.GET_LRS_NODE_GEOMETRY(
    network  IN VARCHAR2,
    node_id  IN NUMBER) RETURN MDSYS.SDO_GEOMETRY;
```

Description

Returns the LRS geometry associated with a node in a spatial LRS network.

Parameters

network
Network name.

node_id
ID number of the node for which to return the geometry.

Usage Notes

None.

Examples

The following example returns the LRS geometry associated with the node whose node ID is 3 in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_LRS_NODE_GEOMETRY('ROADS_NETWORK', 3) FROM DUAL;

SDO_NET.GET_LRS_NODE_GEOMETRY('ROADS_NETWORK',3)(SDO_GTYPE, SDO_SRID, SDO_POINT(
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(8, 4, NULL), NULL, NULL)
```

SDO_NET.GET_LRS_TABLE_NAME

Format

```
SDO_NET.GET_LRS_TABLE_NAME(
    network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns the name of the table containing LRS geometries in a spatial LRS network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the LRS_TABLE_NAME column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the table that contains LRS geometries for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_LRS_TABLE_NAME('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_LRS_TABLE_NAME('ROADS_NETWORK')
--------------------------------------------------------------------------------
ROADS
```

SDO_NET.GET_NETWORK_TYPE

Format

```
SDO_NET.GET_NETWORK_TYPE(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns the network type.

Parameters

network
Network name.

Usage Notes

This function returns the value of the NETWORK_TYPE column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the network type for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NETWORK_TYPE('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_NETWORK_TYPE('ROADS_NETWORK')
--------------------------------------------------------------------------------
Roadways
```

SDO_NET.GET_NO_OF_HIERARCHY_LEVELS

Format

```
SDO_NET.GET_NO_OF_HIERARCHY_LEVELS(
     network  IN VARCHAR2) RETURN NUMBER;
```

Description

Returns the number of hierarchy levels for a network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the NO_OF_HIERARCHY_LEVELS column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

For an explanation of network hierarchy, see Section 6.5.

Examples

The following example returns the number of hierarchy levels for the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NO_OF_HIERARCHY_LEVELS('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_NO_OF_HIERARCHY_LEVELS('ROADS_NETWORK')
---------------------------------------------------
                                                  1
```

SDO_NET.GET_NO_OF_LINKS

Format

```
SDO_NET.GET_NO_OF_LINKS(
    network  IN VARCHAR2) RETURN NUMBER;
```

or

```
SDO_NET.GET_NO_OF_LINKS(
    network  IN VARCHAR2,
    hierarchy_id  IN NUMBER) RETURN NUMBER;
```

Description

Returns the number of links for a network or a hierarchy level in a network.

Parameters

network
Network name.

hierarchy_id
Hierarchy level number for which to return the number of links.

Usage Notes

None.

Examples

The following example returns the number of links in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NO_OF_LINKS('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_NO_OF_LINKS('ROADS_NETWORK')
----------------------------------------
                                      10
```

SDO_NET.GET_NO_OF_NODES

Format

```
SDO_NET.GET_NO_OF_NODES(
    network  IN VARCHAR2) RETURN NUMBER;
```

or

```
SDO_NET.GET_NO_OF_NODES(
    network  IN VARCHAR2,
    hierarchy_id  IN NUMBER) RETURN NUMBER;
```

Description

Returns the number of nodes for a network or a hierarchy level in a network.

Parameters

network
Network name.

hierarchy_id
Hierarchy level number for which to return the number of nodes.

Usage Notes

For information about nodes and related concepts, see Section 6.3.

Examples

The following example returns the number of nodes in the network named ROADS_NETWORK.

```
SELECT SDO_NET.GET_NO_OF_NODES('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_NO_OF_NODES('ROADS_NETWORK')
----------------------------------------
                                       8
```

SDO_NET.GET_NODE_DEGREE

Format

```
SDO_NET.GET_NODE_DEGREE(
    network  IN VARCHAR2,
    node_id  IN NUMBER) RETURN NUMBER;
```

Description

Returns the number of links to a node.

Parameters

network
Network name.

node_id
Node ID of the node for which to return the number of links.

Usage Notes

For information about node degree and related network data model concepts, see Section 6.3.

Examples

The following example returns the number of links to the node whose node ID is 3 in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NODE_DEGREE('ROADS_NETWORK', 3) FROM DUAL;

SDO_NET.GET_NODE_DEGREE('ROADS_NETWORK',3)
------------------------------------------
                                         3
```

SDO_NET.GET_NODE_GEOM_COLUMN

Format

```
SDO_NET.GET_NODE_GEOM_COLUMN(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns the name of the geometry column for nodes in a spatial network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the NODE_GEOM_COLUMN column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the geometry column for nodes in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NODE_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_NODE_GEOM_COLUMN('ROADS_NETWORK')
--------------------------------------------------------------------------------
NODE_GEOMETRY
```

SDO_NET.GET_NODE_GEOMETRY

Format

SDO_NET.GET_NODE_GEOMETRY(

network IN VARCHAR2,

node_id IN NUMBER) RETURN MDSYS.SDO_GEOMETRY;

Description

Returns the LRS geometry associated with a node in a spatial network.

Parameters

network
Network name.

node_id
ID number of the node for which to return the geometry.

Usage Notes

None.

Examples

The following example returns the geometry associated with the node whose node ID is 3 in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NODE_GEOMETRY('ROADS_NETWORK', 3) FROM DUAL;

SDO_NET.GET_NODE_GEOMETRY('ROADS_NETWORK',3)(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(8, 4, NULL), NULL, NULL)
```

SDO_NET.GET_NODE_IN_DEGREE

Format

```
SDO_NET.GET_NODE_IN_DEGREE(
     network  IN VARCHAR2,
     node_id  IN NUMBER) RETURN NUMBER;
```

Description

Returns the number of inbound links to a node.

Parameters

network
Network name.

node_id
Node ID of the node for which to return the number of inbound links.

Usage Notes

For information about node degree and related network data model concepts, see Section 6.3.

Examples

The following example returns the number of inbound links to the node whose node ID is 3 in the network named ROADS_NETWORK.

```
SELECT SDO_NET.GET_NODE_IN_DEGREE('ROADS_NETWORK', 3) FROM DUAL;

SDO_NET.GET_NODE_IN_DEGREE('ROADS_NETWORK',3)
---------------------------------------------
                                            1
```

SDO_NET.GET_NODE_OUT_DEGREE

Format

```
SDO_NET.GET_NODE_OUT_DEGREE(
    network  IN VARCHAR2,
    node_id  IN NUMBER) RETURN NUMBER;
```

Description

Returns the number of outbound links from a node.

Parameters

network
Network name.

node_id
Node ID of the node for which to return the number of outbound links.

Usage Notes

For information about node degree and related network data model concepts, see Section 6.3.

Examples

The following example returns the number of outbound links from the node whose node ID is 3 in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NODE_OUT_DEGREE('ROADS_NETWORK', 3) FROM DUAL;

SDO_NET.GET_NODE_OUT_DEGREE('ROADS_NETWORK',3)
----------------------------------------------
                                             2
```

SDO_NET.GET_NODE_TABLE_NAME

Format

```
SDO_NET.GET_NODE_TABLE_NAME(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns the name of the table that contains the nodes in a spatial network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the NODE_TABLE_NAME column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the table that contains the nodes in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_NODE_TABLE_NAME('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_NODE_TABLE_NAME('ROADS_NETWORK')
--------------------------------------------------------------------------------
ROADS_NODES
```

SDO_NET.GET_OUT_LINKS

Format

SDO_NET.GET_OUT_LINKS(

network IN VARCHAR2,

node_id IN NUMBER) RETURN SDO_NUMBER_ARRAY;

Description

Returns an array of link ID numbers of the outbound links from a node.

Parameters

network
Network name.

node_id
ID of the node for which to return the array of outbound links.

Usage Notes

For information about outbound links and related network data model concepts, see Section 6.3.

Examples

The following example returns an array of link ID numbers of the outbound links from the node whose node ID is 3 in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_OUT_LINKS('ROADS_NETWORK', 3) FROM DUAL;

SDO_NET.GET_OUT_LINKS('ROADS_NETWORK',3)
--------------------------------------------------------------------------------
SDO_NUMBER_ARRAY(103, 201)
```

SDO_NET.GET_PATH_GEOM_COLUMN

Format

SDO_NET.GET_PATH_GEOM_COLUMN(

network IN VARCHAR2) RETURN VARCHAR2;

Description

Returns the name of the geometry column for paths in a spatial network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the PATH_GEOM_COLUMN column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the geometry column for paths in the network named ROADS_NETWORK.

```
SELECT SDO_NET.GET_PATH_GEOM_COLUMN('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_PATH_GEOM_COLUMN('ROADS_NETWORK')
--------------------------------------------------------------------------------
PATH_GEOMETRY
```

SDO_NET.GET_PATH_TABLE_NAME

Format

SDO_NET.GET_PATH_TABLE_NAME(

    network IN VARCHAR2) RETURN VARCHAR2;

Description

Returns the name of the table that contains the paths in a spatial network.

Parameters

network
Network name.

Usage Notes

This function returns the value of the PATH_TABLE_NAME column for the network in the USER_SDO_NETWORK_METADATA view (see Table 6–5 in Section 6.7.1).

Examples

The following example returns the name of the table that contains the paths in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.GET_PATH_TABLE_NAME('ROADS_NETWORK') FROM DUAL;

SDO_NET.GET_PATH_TABLE_NAME('ROADS_NETWORK')
--------------------------------------------------------------------------------
ROADS_PATHS
```

SDO_NET.IS_HIERARCHICAL

Format

```
SDO_NET.IS_HIERARCHICAL(
    network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the network has more than one level of hierarchy; returns `FALSE` if the network does not have more than one level of hierarchy.

Parameters

network
Network name.

Usage Notes

For an explanation of network hierarchy, see Section 6.5.

Examples

The following example checks if the network named `ROADS_NETWORK` has more than one level of hierarchy.

```
SELECT SDO_NET.IS_HIERARCHICAL('ROADS_NETWORK') FROM DUAL;

SDO_NET.IS_HIERARCHICAL('ROADS_NETWORK')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.IS_LOGICAL

Format

SDO_NET.IS_LOGICAL(

    network IN VARCHAR2) RETURN VARCHAR2;

Description

Returns `TRUE` if the network is a logical network; returns `FALSE` if the network is not a logical network.

Parameters

network
Network name.

Usage Notes

A network can be a spatial network or a logical network, as explained in Section 6.3.

Examples

The following example checks if the network named `ROADS_NETWORK` is a logical network.

```
SELECT SDO_NET.IS_LOGICAL('ROADS_NETWORK') FROM DUAL;

SDO_NET.IS_LOGICAL('ROADS_NETWORK')
--------------------------------------------------------------------------------
FALSE
```

SDO_NET.IS_SPATIAL

Format

```
SDO_NET.IS_SPATIAL(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the network is a spatial network; returns `FALSE` if the network is not a spatial network.

Parameters

network
Network name.

Usage Notes

A network can be a spatial network or a logical network, as explained in Section 6.3.

You can further check for the geometry type of a spatial network by using the following functions: SDO_NET.LRS_GEOMETRY_NETWORK, SDO_NET.SDO_GEOMETRY_NETWORK, and SDO_NET.TOPO_GEOMETRY_NETWORK.

Examples

The following example checks if the network named `ROADS_NETWORK` is a spatial network.

```
SELECT SDO_NET.IS_SPATIAL('ROADS_NETWORK') FROM DUAL;

SDO_NET.IS_SPATIAL('ROADS_NETWORK')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.LRS_GEOMETRY_NETWORK

Format

```
SDO_NET.LRS_GEOMETRY_NETWORK(
   network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the network is a spatial network containing LRS geometries; returns `FALSE` if the network is not a spatial network containing LRS geometries.

Parameters

network
Network name.

Usage Notes

A network contains LRS geometries if the GEOMETRY_TYPE column in its entry in the USER_SDO_NETWORK_METADATA view contains the value `LRS_GEOMETRY`. (The USER_SDO_NETWORK_METADATA view is explained in Section 6.7.1.)

Examples

The following example checks if the network named `ROADS_NETWORK` is a spatial network containing LRS geometries.

```
SELECT SDO_NET.LRS_GEOMETRY_NETWORK('ROADS_NETWORK') FROM DUAL;

SDO_NET.LRS_GEOMETRY_NETWORK('ROADS_NETWORK')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.NETWORK_EXISTS

Format

```
SDO_NET.NETWORK_EXISTS(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the network exists; returns `FALSE` if the network does not exist.

Parameters

network
Network name.

Usage Notes

If you drop a network (using the SDO_NET.DROP_NETWORK procedure), the network no longer exists.

Examples

The following example checks if the network named `ROADS_NETWORK` exists.

```
SELECT SDO_NET.NETWORK_EXISTS('ROADS_NETWORK') FROM DUAL;

SDO_NET.NETWORK_EXISTS('ROADS_NETWORK')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.SDO_GEOMETRY_NETWORK

Format

```
SDO_NET.SDO_GEOMETRY_NETWORK(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the network is a spatial network containing SDO geometries (spatial geometries without measure information); returns `FALSE` if the network is not a spatial network containing SDO geometries.

Parameters

network
Network name.

Usage Notes

A network contains SDO geometries if the GEOMETRY_TYPE column in its entry in the USER_SDO_NETWORK_METADATA view contains the value `SDO_GEOMETRY`. (The USER_SDO_NETWORK_METADATA view is explained in Section 6.7.1.)

Examples

The following example checks if the network named `ROADS_NETWORK` is a spatial network containing SDO geometries.

```
SELECT SDO_NET.SDO_GEOMETRY_NETWORK('ROADS_NETWORK') FROM DUAL;

SDO_NET.SDO_GEOMETRY_NETWORK('ROADS_NETWORK')
--------------------------------------------------------------------------------
FALSE
```

SDO_NET.TOPO_GEOMETRY_NETWORK

Format

```
SDO_NET.TOPO_GEOMETRY_NETWORK(
    network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the network is a spatial network containing SDO_TOPO_GEOMETRY (topology geometry) objects; returns `FALSE` if the network is not a spatial network containing SDO_TOPO_GEOMETRY objects.

Parameters

network
Network name.

Usage Notes

A network contains SDO_TOPO_GEOMETRY objects if the GEOMETRY_TYPE column in its entry in the USER_SDO_NETWORK_METADATA view contains the value `TOPO_GEOMETRY`. (The USER_SDO_NETWORK_METADATA view is explained in Section 6.7.1.)

Examples

The following example checks if the network named `ROADS_NETWORK` is a spatial network containing SDO_TOPO_GEOMETRY objects.

```
SELECT SDO_NET.TOPO_GEOMETRY_NETWORK('ROADS_NETWORK') FROM DUAL;

SDO_NET.TOPO_GEOMETRY_NETWORK('ROADS_NETWORK')
--------------------------------------------------------------------------------
FALSE
```

SDO_NET.VALIDATE_LINK_SCHEMA

Format

```
SDO_NET.VALIDATE_LINK_SCHEMA(
    network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the metadata relating to links in a network is valid; returns `FALSE` if the metadata relating to links in a network is not valid.

Parameters

network
Network name.

Usage Notes

This function checks the following for validity: table name, geometry column, and cost column for spatial networks; measure-related information for LRS networks; topology-related information for topology networks; and hierarchy-related information for hierarchical networks.

Examples

The following example checks the validity of the metadata related to links in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.VALIDATE_LINK_SCHEMA('ROADS_NETWORK') FROM DUAL;

SDO_NET.VALIDATE_LINK_SCHEMA('ROADS_NETWORK')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.VALIDATE_LRS_SCHEMA

Format

SDO_NET.VALIDATE_LRS_SCHEMA(

  network IN VARCHAR2) RETURN VARCHAR2;

Description

Returns `TRUE` if the metadata relating to LRS information in a network is valid; returns `FALSE` if the metadata relating to LRS information in a network is not valid.

Parameters

network
Network name.

Usage Notes

None.

Examples

The following example checks the validity of the metadata related to LRS information in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.VALIDATE_LRS_SCHEMA('ROADS_NETWORK') FROM DUAL;

SDO_NET.VALIDATE_LRS_SCHEMA('ROADS_NETWORK')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.VALIDATE_NETWORK

Format

```
SDO_NET.VALIDATE_NETWORK(
    network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the network is valid; returns `FALSE` if the network is not valid.

Parameters

network
Network name.

Usage Notes

This function checks for the following, and returns `FALSE` if one or more are not true:

- The network exists.
- The node and link tables for the network exist, and they contain the required columns.
- For an LRS geometry network, the LRS table exists and contains the required columns.
- For a spatial network, columns for the node and path geometries exist and have spatial indexes defined on them.

Examples

The following example validates the network named `LOG_NET1`.

```
SELECT SDO_NET.VALIDATE_NETWORK('LOG_NET1') FROM DUAL;

SDO_NET.VALIDATE_NETWORK('LOG_NET1')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.VALIDATE_NODE_SCHEMA

Format

```
SDO_NET.VALIDATE_NODE_SCHEMA(
     network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the metadata relating to nodes in a network is valid; returns `FALSE` if the metadata relating to nodes in a network is not valid.

Parameters

network
Network name.

Usage Notes

This function checks the following for validity: table name, geometry column, and cost column for spatial networks; measure-related information for LRS networks; topology-related information for topology networks; and hierarchy-related information for hierarchical networks.

Examples

The following example checks the validity of the metadata related to nodes in the network named `LOG_NET1`.

```
SELECT SDO_NET.VALIDATE_NODE_SCHEMA('LOG_NET1') FROM DUAL;

SDO_NET.VALIDATE_NODE_SCHEMA('LOG_NET1')
--------------------------------------------------------------------------------
TRUE
```

SDO_NET.VALIDATE_PATH_SCHEMA

Format

```
SDO_NET.VALIDATE_PATH_SCHEMA(
    network  IN VARCHAR2) RETURN VARCHAR2;
```

Description

Returns `TRUE` if the metadata relating to paths in a network is valid; returns `FALSE` if the metadata relating to paths in a network is not valid.

Parameters

network
Network name.

Usage Notes

This function checks the following for validity: table name, geometry column, and cost column for spatial networks; measure-related information for LRS networks; topology-related information for topology networks; and hierarchy-related information for hierarchical networks.

Examples

The following example checks the validity of the metadata related to paths in the network named `ROADS_NETWORK`.

```
SELECT SDO_NET.VALIDATE_PATH_SCHEMA('ROADS_NETWORK') FROM DUAL;

SDO_NET.VALIDATE_PATH_SCHEMA('ROADS_NETWORK')
--------------------------------------------------------------------------------
TRUE
```

Index