Report on

'Pro Oracle Spatial for Oracle Database 11g'

UNIT: FIT 5091– Research Topic

Present By:

Qi Mao (ID:19691556)

EMAIL: qmao1@student.monash.edu.au

PHONE: 0413 646 352

Semester 1, 2009

Supervisor: DR. DAVID TANIAR

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Spatial Information Management

Many organizations realized the spatial information has become the vast majority component of their information assets, which may include the location, customers, shipments, facilities, personnel, competitors, and so on. Therefore, the ability to use this information suitably is fundamental to reducing operational costs, optimizing production efficiency, increasing productivity and the quality of service. Obvious, managing and exploiting spatial information is plentiful, and it is importance in business applications for business operations and decision-making. Therefore, the Geographical Information Systems (GIS) are extensively used to manage spatial data and exploit spatial intelligence, which is often used as stand-alone systems. Moreover, Oracle Spatial has become to be one of the most established solutions for providing spatial intelligence to databases. Importantly, Oracle Spatial manages spatial data as normal data type, and makes the principle easy for experienced database developers and architects, which brings advantage compacts into spatial information management. Furthermore, Oracle Spatial supports MapViewer technology to create and integrate maps very easy in business applications.

1.2 Using Spatial in Industries

Spatial information is used in different industries to conduct valuable business analyses for establishing business operations and decision-making. Therefore, the representative of a vast class of uses for spatial information including Banking and Finance, Telecommunications, Government, Retail and Wireless data services. For instead, it can help determine new marketing campaigns, opening of new stores, and discontinuation of poorly located stores, identify efficient home-delivery schedules, change the stores' product portfolios, and so on. It considers the following options:

- *Identify customers that are close to a competitor store:* help designing a specific marketing campaign strategy these customers
- *Optimize the distribution network:* by counting the number of customers who are located within, overloaded or underutilized a certain distance from a distribution center, it able to help with redesign the distribution network
- *Identify routes from delivery sites to customer location:* unable plan each delivery to serve multiple customers and save money
- *Superimpose the location of stores on a population map:* unable check whether the store locations are appropriate using additional demographic data
- *Visualize table data and analysis results as maps:* produce rich visual material for communication and decision making

- *Integrate these maps with existing applications:* with CRM system, the location information and analysis can promote effective customer relations

To perform these types of analyses, basic location information for customers, delivery sites, and competitors need to be stored in the database. Besides, the analysis in visualization and analysis may require some other additional information, such as street networks, rivers, city and state boundaries, and so on.

1.3 Benefits of Oracle Spatial

The basic advantages of using Oracle Spatial can be summarized as follows:

- *Reduces dual architectures:* all data can be stored in the same way
- *Data storage:* all types of data (text, maps, and multimedia) are stored together, instead of each type being stored separately
- *Uses SQL:* no need for specific languages to handle spatial data
- *A SDO_GEOMETRY data type:* is essentially equivalent to the spatial types in the OGC and SQL/MM standards
- *SQL/MM "well-known" formats:* the SQL/MM specifications can easily store the data in Oracle Spatial for specifying spatial data, and vice versa, without the need for third-party converters
- *A de facto standard:* is fully supported by the world's leading geospatial data, tools, and applications vendors, including NAVTEQ, Tele Atlas, Digital Globe, 1Spatial, Autodesk, Bentley, eSpatial, ESRI, GE Energy/Smallworld, Intergraph, Leica Geosystems, Manifold, PCI Geomatics, Pitney/Bowes/MapInfo, Safe Software, Skyline, and many others.
- *Powerful features:* provides scalability, integrity, security, recoverability, and advanced user management features for handling spatial data
- *Powerful maintenance:* eliminates the need for specific tools and skills for operating spatial data, and maintain the spatial data infrastructure
- *Through the application server:* allows almost any application to benefit from the availability of spatial information and intelligence, reducing the costs and complexity of spatial applications
- *The benefits to grid computing:* provides substantial cost savings and easier maintenance of the database structures for large organizations to manage their data assets
- *Powerful visualization:* no need to rely a separate visualization tools for many applications

1.4 Managing and Analyzing Spatial Data

Oracle Spatial contains a number of SQL operators and functions, and some of typically spatial operations are listed in below:

Storage of Spatial Data

- Storing the data in an appropriate form in the database
- Inserting, deleting, and updating these types of spatial data in the database

Analysis of Vector Spatial Data

- Within-distance operation: identifies all spatial data within a specified distance of a query location
- Contains operation: identifies all spatial data that contain a specified query location
- Nearest-neighbor operation: identifies all spatial data closest to a query location
- Distance operation: computes the distance between two spatial objects
- Buffer operation: constructs buffer zones around spatial data
- Overlay operation: overlays different layers of spatial data
- Visualization operation: presents spatial data using maps

Analysis of Network Data

- Most spatial data like road networks can be represented as network data (in addition to vector data), which can use network proximity to perform the preceding analysis.

• Storing Spatial Data in a Database

Most spatial databases use a spatial data type to store spatial information as points, lines, polygons, and other types of vector representations in a database, which is referred as the geometry data type. It allows user to add columns of type geometry to a database table in order to store spatial objects. Moreover, the system may also have a network type for modeling road networks.

Particularly, the vector data is stored in one or multiple tables, and it usually distinguished as following:

- *Points* (for example, the plots for sale): requires only x, y coordinates (or x, y, z if 3D is considered)
- *Lines* (for example, roads): requires a start coordinate, an end coordinate, and a certain number of intermediate coordinates
- *Polygons* (for example, a residential area): described by closed lines

• Spatial Analysis

In general, performing spatial analysis is an important intention of using oracle spatial. It can obtain meaningful information from data once it is stored in the appropriate form in a database.

Assuming we have the data stored in a database, so below shows an example of using three common spatial operations for a site selection:

Select:

- Step 1 (to select areas where the attribute was a certain value)

```
-- to identify available plots of land for which a construction permit can be obtained for a shopping mall
```

```
SELECT AREA_NAME, AREA_GEOMETRY
FROM M1
WHERE LAND_USE_TYPE='COMMERCIAL'
```

- Step 2 (to select large sites from the sites for sale)

```
-- to identify available sites whose size is larger than a certain value
SELECT SITE_NAME, SITE_GEOMETRY
FROM M2
WHERE SITE_PLOT_AREA > <some value>
```

- Step 4 (to select major roads from the road network)

```
-- to identify major roads
SELECT ROAD_NAME, ROAD_GEOMETRY
FROM M4
WHERE ROAD_TYPE='MAJOR ROAD'
```

Overlay:

- Step 5 (large sites in commercial areas)

```
-- Use the contains operator to identify the sites selected in step 2 that are within areas selected in step 1. You could also achieve this in one step starting directly from M1 and M2:
SELECT SITE_NAME, SITE_GEOMETRY
FROM M2 S, M1 L
WHERE CONTAINS(L.AREA_GEOMETRY, S.SITE_GEOMETRY)='TRUE'
AND L.LAND_USE_TYPE= 'COMMERCIAL'
AND S.SITE_AREA > <some value>;
```

- Step 7 (sites away from risk areas)

```
-- Use contains to identify sites selected in step 5 that are outside the flood-prone areas identified in step 3.
```

- Step 8 (sites within highly accessible areas)

```
-- Use contains to identify safe sites selected in step 7 that are within the zones of easy accessibility created in step 6.
```

Buffer:

- Step 3 (areas subject to flood risk)

```
-- to create a buffer of 1 kilometer around the named river
SELECT BUFFER(RIVER_GEOMETRY, 1, 'unit=km')
FROM M3
WHERE RIVER_NAME= <river_in_question>
```

- Step 6 (high accessibility areas)

```
As in step 3, use the buffer function to create a buffer of a certain size around the major roads.
```

1.5 Report Outline

The purpose of this report is to introduce and experiment the Oracle Spatial features for geometry data analyzing. Particularly, all examples are used for expiations, demonstrations and practice in this report that are referred to the 'Pro Oracle Spatial for Oracle Database 11g'. Furthermore, the contents and examples in this paper are operated in Oracle Database 11g with SQL*Plus. In addition, this paper contains ten sections that can be described as three major parts that are showing as below:

- **Overview of Oracle Spatial**

  0.1 Introduction:

  This section provides a basic concept of Oracle Spatial which including:
  - Spatial information management
  - Spatial in the industries
  - Benefits of using Oracle Spatial
  - Spatial Data management and analysis

  2.0 Overview of Oracle Spatial:

  This section provides an overview of Oracle Spatial technology suite that enables spatial information management, and it covers:
  - Storage using SDO_GEOMETRY
  - Analysis and querying using spatial operators
  - Installation of Oracle Spatial

  3.0 Location-Enabling Your Applications:

  This section used an e-business application to experiment how to augment existing application tables with location information. It contains three main topics, which are:
  - Storing geographic data in Oracle tables
  - Populating appropriate metadata tables to enable spatial processing on spatial tables
  - Detailed populating of metadata

- **Basic Spatial**

  4.0 The SDO_GEOMETRY Data Type:

  This section focuses on the storage and modeling of location information using the SDO_GEOMETRY data type in Oracle, which covers:
  - Overview of SDO_GEOMETRY data type
  - Using SDO_GEOMETRY for the simple Two-Dimensional geometry
  - Using SDO_GEOMETRY for the complex Two-Dimensional geometry
  - Using SDO_GEOMETRY for the Three-Dimensional geometry

5.0 Loading, Transporting, and Validating Spatial Data:

This section examined different ways to populate Oracle tables that contain SDO_GEOMETRY columns content, which covers five major parts:

- Loading data into and out of SDO_GEOMETRY columns
- Using the Oracle utilities such as Import/Export and transportable tablespaces
- Converting SDO_GEOMETRY data to GML format
- Performing validation to check for conformity with Oracle Spatial formats
- Creating functions in debugging

6.0 Geocoding:

This section tested the functionality of the geocoder in Oracle Spatial, which includes:

- Introduction of geocoding concepts and the geocoding process
- Setting up a data catalog for enabling geocoding

7.0 Manipulating SDO_GEOMETRY in Application Programs:

This section examines the manipulation of SDO_GEOMETRY types by reading, decoding, constructing, and writing geometries using examples throughout.

- **Spatial & Network Analysis**

8.0 Spatial Indexes and Operators:

In this section, examples are used to experiment the proximity analysis using spatial information, which focus on:

- Concepts and creating spatial indexes
- Syntax and semantics of spatial operators
- Advanced features of spatial index

9.0 Geometry Processing Functions:

This section examines several main geometry processing functions, which includes:

- Buffering functions
- Relationship analysis functions
- Geometry combination functions
- Geometric analysis functions
- Aggregate functions

10.0 Network Modeling:

This section introduces and tests a network concept for modeling spatial data using SQL, which covers:

- Defining networks
- Analyzing networks

2.0 OVERVIEW OF ORACLE SPATIAL

The Oracle Spatial technology suite enables spatial information management inside Oracle, and it contents the various components, which enables storing and analyzing spatial data in the database. This section examined the functionality of different components in Oracle Spatial that included:

- the data type for storing spatial data
- new operators and functions to perform spatial query and analysis

2.1 Starting with Oracle Spatial

- **Data Model: Storing Spatial Data**

  The spatial information is specified using two components:

  - Original location: specifies where the data is located (e.g. a two-, three-, or four-dimensional space)
  - Geometric shape: specifies the geometric structure of the data. (e.g. Point, line, and polygon)

  The SDO_GEOMETRY is represent as an Oracle object data type that can capture the location and shape information of data rows in a table. It can model different shapes such as points, lines, polygons, and combination shapes in spatial data of most spatial applications.

- **Location Enabling**

  In this part, we would discuss the creation of SDO_GEOMETRY for storing spatial data in Oracle, and the method to populate spatial tables with data.

  Table Creation

  The SDO_GEOMETRY data type is used in create tables to store locations.

  For example, creating the us_restaurants_new Table:

```
SQL> CREATE TABLE us_restaurants_new
  2  (
  3     id NUMBER,
  4     poi_name VARCHAR2(32),
  5     location SDO_GEOMETRY -- New column to store locations
  6  );

  Table created.
```

  Data Insertion

  Actually, the SDO_GEOMETRY is just like any other object type, which can populate an SDO_GEOMETRY column using the corresponding object constructor.

For example, insert a location of (–87, 38) for a Pizza Hut restaurant into the us_restaurants table:

```
SQL> INSERT INTO us_restaurants_new VALUES
  2  (
  3   1,
  4   'PIZZA HUT',
  5   SDO_GEOMETRY
  6   (
  7    2001, -- SDO_GTYPE attribute: "2" in 2001 specifies dimensionality is 2.
  8    NULL, -- other fields are set to NULL.
  9    SDO_POINT_TYPE -- Specifies the coordinates of the point
 10    (
 11    -87, -- first ordinate, i.e., value in longitude dimension
 12    38, -- second ordinate, i.e., value in latitude dimension
 13    NULL -- third ordinate, if any
 14    ),
 15    NULL,
 16    NULL
 17   )
 18  );

1 row created.
```

In this example:

- '2001' specifies it is a two-dimensional point geometry (a line would be represented by 2002, a polygon by 2003, and a collection by 2004)
- the location of this point in the SDO_POINT attribute using SDO_POINT_TYPE constructor

Spatial Information Conversion

In the situation when the spatial information is not explicitly available as coordinates in the application, the address data can be converted into an SDO_GEOMETRY object using the geocoder component.

For example, use the geocoder to obtain the coordinates in the form of an SDO_GEOMETRY object for the address '3746 CONNECTICUT AVE NW' in Washington, D.C.

```
SQL> SELECT
  2  SDO_GCDR.GEOCODE_AS_GEOMETRY
  3  (
  4   'test', -- Spatial schema storing the geocoder data
  5   SDO_KEYWORDARRAY -- Object combining different address components
  6   (
  7   '3746 CONNECTICUT AVE NW',
  8   'WASHINGTON, DC 20008'
  9   ),
```

```
 10    'US' -- Name of the country
 11   ) geom
 12   FROM DUAL ;

GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-77.060283, 38.9387083, NULL), NULL,
NULL)
```

In this example:

- 'test' is the schema
- the second argument specifies an SDO_KEYWORDARRAY object (constructed out of the street component: '3746 Connecticut Ave NW', the city/ZIP code component: 'Washington, DC 20008'), composed from different components of an address
- the 'US' dataset is being used to geocode the specified street address and the function returns an SDO_GEOMETRY type with the longitude set to −77.060283 and the latitude set to 38.9387083

- **Query and Analysis**

The query and analysis component provides the core functionality to perform spatial queries and analysis for spatial geometries, which including two subcomponents: a Geometry Engine and an Index Engine.

The Geometry Engine

The Geometry Engine provides functions to analyze, compare, and manipulate geometries.

For instance, use the Geometry Engine functionality to identify the nearest five restaurants on I-795 in the greater Washington, D.C., area.

```
SQL> SELECT poi_name
  2  FROM
  3  (
  4   SELECT poi_name,
  5   SDO_GEOM.SDO_DISTANCE(P.location, I.geom, 0.5) distance
  6   FROM us_interstates I, us_restaurants P
  7   WHERE I.interstate = 'I795'
  8   ORDER BY distance
  9  )
 10  WHERE ROWNUM <= 5;

POI_NAME
----------------------------------------------------------------
PIZZA BOLI'S
BLAIR MANSION INN DINNER THEATER
```

```
KFC
CHINA HUT
PIZZA HUT


5 rows selected.
```

In this example:

- the inner SELECT clause computes the distance between I-795 and each restaurant row of the us_restaurants table using the Geometry Engine function 'SDO_GEOM.SDO_DISTANCE'
- the ORDER BY clause sorts the results in ascending order of distance
- the outer SELECT statement selects the five nearest restaurants

The Index Engine

The spatial Index Engine can be used to speed up query processing by minimizing the processing overhead.

For example, creating an Index on Locations (SDO_GEOMETRY Column) of Restaurants:

```
-- drop the existing index
SQL> DROP INDEX us_restaurants_sidx; -- drop current index for us_restaurants

Index dropped.

-- recreate the index on location of restaurants table
SQL> CREATE INDEX us_restaurants_sidx ON us_restaurants(location)
  2  INDEXTYPE IS mdsys.spatial_index;

Index created.
```

In this example:

- the clause INDEXTYPE tells the database to create a spatial index on the location (SDO_GEOMETRY) column of the us_restaurants table

Examples of testing the spatial index that have created:

Finding the Five nearest Restaurants on I-795 Using the Spatial Index:

```
SQL> SELECT poi_name
  2  FROM us_interstates I, us_restaurants P
  3  WHERE I.interstate = 'I795'
  4  AND SDO_NN(P.location, I.geom) ='TRUE'
  5  AND ROWNUM <= 5;

POI_NAME
----------------------------------------------------------------
PIZZA BOLI'S
BLAIR MANSION INN DINNER THEATER
KFC
```

```
CHINA HUT
PIZZA HUT


5 rows selected.
```

In this example:

- the SDO_NN function is a index-based operator which returns these rows in order of proximity to the I-795 geometry

Identifying All Restaurants in a 50 km Radius around I-795

```
SQL> SELECT POI_NAME
  2  FROM us_interstates I, us_restaurants P
  3  WHERE
  4  SDO_ANYINTERACT
  5  (
  6   P.location,
  7   SDO_GEOM.SDO_BUFFER(I.geom, 50, 0.5, 'UNIT=KM')
  8  ) ='TRUE'
  9  AND I.interstate='I795' ;

POI_NAME
----------------------------------------------------------------
SPICY DELIGHT
PHILLY'S STEAK EXPRESS
MCDONALD'S
PIZZA HUT
CHINA HUT
KFC
BLAIR MANSION INN DINNER THEATER
PIZZA BOLI'S


8 rows selected.
```

In this example:

- the *SDO_ANYINTERACT* function is an index-based operator which identifies all rows of us_restaurants where the locations intersect with the geometry passed in as the second parameter
- the *SDO_BUFFER* function generates and returns a 50 km buffer around the I-795 geometry

2.2 Oracle Spatial Installation

- **Installing Oracle Spatial in the Database**
  Oracle Spatial is automatically installed with the Standard Edition or Enterprise Edition of an Oracle Database Server. All Spatial data types, views, packages, and functions are installed as part of a schema called MDSYS, and the existing MDSYS account can be used to verify whether Spatial has been installed properly.
  Verifying That a Spatial Install Is Successful:

```
SQL> SELECT COMP_NAME, STATUS
  2  FROM DBA_REGISTRY
  3  WHERE COMP_NAME = 'Spatial';

COMP_NAME                                                        STATUS
---------------------------------------------------------------- --------------------
Spatial                                                          VALID
```

- **Understanding a Spatial Install**
  Checking for Invalid Objects in a Spatial Installation:

```
SQL> SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
  2  FROM ALL_OBJECTS
  3  WHERE OWNER='MDSYS' AND STATUS <> 'VALID'
  4  ORDER BY OBJECT_NAME;

no rows selected
```

  In this example:
  - If any rows returned, contact Oracle Support for troubleshooting help

- **Checking the Version of a Spatial Install**
  Checking for the Version of a Spatial Install

```
SQL> SELECT SDO_VERSION FROM DUAL;

SDO_VERSION
--------------------------------------------------------------------------------
11.1.0.6.0
```

  In this example:
  - The version of this a Spatial Install is 11.1.0.6.0

3.0 LOCATION-ENABLING

This section covered the major steps required to location-enable business applications (stores information such as its branches, customers, competitors, and suppliers), which includes:

- Designing and creating tables to store application-specific data
- Designing and creating tables to store geographic data
- Defining metadata for each spatial layer both in the application-specific and the

The spatial application data and the geographic data are stored using an SDO_GEOMETRY object.

3.1 Adding Location Information to Tables

The application data can be categorized into two types:

- *Application-specific tables:* contain information that is specific to the application (product and customer information, and so on)
- *Geographic tables:* contain geographic data which is independent of the application and contains columns to store explicit spatial information for street networks, city boundaries, and so on

- **Application-Specific Data**

  Assuming the application includes five tables, a products table, a customers table, a suppliers table, a branches table and a competitors table. Table can be created with appropriate attributes and populated using SQL INSERT statements. For example, creating customer table and populate the data.

  Creating the customers Table:

```
SQL> CREATE TABLE customers
  2  (
  3  id NUMBER,
  4  datasrc_id NUMBER,
  5  name VARCHAR2(35),
  6  category VARCHAR2(30),
  7  street_number VARCHAR2(5),
  8  street_name VARCHAR2(60),
  9  city VARCHAR2(32),
 10  postal_code VARCHAR2(16),
 11  state VARCHAR2(32),
 12  phone_number VARCHAR2(15),
 13  customer_grade VARCHAR2(15)
 14  );

Table created.
```

Populating the customers Table:

```
SQL> INSERT INTO customers VALUES
(
1, -- id
1, -- datasrc_id
'Pizza Hut' , -- name
'Restaurant', -- restaurant
'134', -- street_number
'12TH STREET', -- street_name
'WASHINGTON', -- city
'20003', -- postal_code
'DC', -- state
NULL, -- phone_number
'GOLD' -- customer_grade
);

1 row created.
```

Adding Location to Application-Specific Data

The SDO_GEOMETRY type can be used to store basic location information in the fundamental level.

Below is an example of using ‘alter’ to add location information in the customers table.

Adding a location Column to the customers Table:

```
SQL> ALTER TABLE customers ADD (location SDO_GEOMETRY);

Table altered.
```

Below is an example of checking the data which have been inserted in to the customers table with ID = 1.

Example Address for a Specific Customer in the customers Table:

```
SQL> SELECT street_number, street_name, city, state, postal_code
FROM customers
WHERE id = 1;

STREE STREET_NAME            CITY                           STATE     POSTAL_CODE
----- --------------------------- ---------------------------------------------------------------- -------------------------
134     12TH STREET            WASHINGTON                     DC           20003
```

Oracle Spatial allows using ‘sdo_gcdr.geocode_as_geometry’ function to convert the address (street_number, street_name, city, and postal_code) into a two-dimensional point location on the surface of the earth. For example, geocoding sample addresses to obtain explicit spatial information.

```
SQL> UPDATE customers
SET location =
```

```
  3  SDO_GCDR.GEOCODE_AS_GEOMETRY
  4  (
  5  'test',
  6  SDO_KEYWORDARRAY
  7  (
  8   street_number || ' ' || street_name, -- add whitespace between street_number and
street_name
  9   city || ', ' || state || ' ' || postal_code
 10  ),
 11  'US'
 12  ) ;

1 row updated.
```

In this example:

- the *'sdo_keywordarray'* object constructed out of the address components street_number, street_name, city, and postal_code

Using simple select the location column from the customers table to examine what the location information looks like.

Geocoded location Column in the customers Table:

```
SQL> SELECT location
  2  FROM customers
  3  WHERE id=1;

LOCATION(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-76.99022, 38.888654, NULL), NULL,
NULL)
```

In this example:

- the specified address (street_number='134', street_name='12th ST SE', city='WASHINGTON', and postal_code='20003') translates to an *SDO_GEOMETRY* object with longitude and latitude values of –76.99022 and 38.888654 in the *sdo_point* attribute (instantiated using the SDO_POINT_TYPE object)
- the sdo_gtype value of *'2001'* indicates that the location is a two-dimensional location

Once an SDO_GEOMETRY object is constructed, the queries like insert and update can be used as normal in any other column in an Oracle table. For instance, the location column can be update directly by constructing a geometry object using an SDO_GEOMETRY constructor.

```
SQL> UPDATE customers
  2  SET location =
```

```
  3  SDO_GEOMETRY
  4  (
  5  2001, -- Specify that location is a point
  6  8307, -- Specify coordinate system id
  7  SDO_POINT_TYPE(-77.06, 38.94, NULL), -- Specify coordinates here
  8  NULL,
  9  NULL
 10  )
 11  WHERE id=1;

1 row updated.
```

- **Geographic Data**

  Geographic data is usually available from a variety of sources, including commercial Geographical Information Systems (GIS) vendors and national mapping agencies. It is used to perform more refined analysis (such as routing between two locations or visualization using regional maps), and accurately visualize the locations stored in application tables on a map. Therefore, the geographic data need to be divided more specifically into multiple tables.

  For example, states have different attributes from other geometries that can be created in a separate us_states table.

```
SQL> CREATE TABLE us_states
  2  (
  3  state VARCHAR2(26),
  4  state_abrv VARCHAR2(2),
  5  totpop NUMBER,
  6  landsqmi NUMBER,
  7  poppssqmi NUMBER,
  8  medage NUMBER,
  9  medhhinc NUMBER,
 10  avghhinc NUMBER,
 11  geom SDO_GEOMETRY
 12  );

Table created.
```

More tables creation:

```
-- create table us_counties
SQL> CREATE TABLE us_counties
  2  (
  3  id NUMBER NOT NULL,
  4  county VARCHAR2(31),
  5  state VARCHAR2(30),
  6  state_abrv VARCHAR2(2),
```

```
  7  landsqmi NUMBER,
  8  totpop NUMBER,
  9  poppsqmi NUMBER,
 10  geom SDO_GEOMETRY
 11  );

-- create table us_interstates
SQL> CREATE TABLE us_interstates
  2  (
  3  id NUMBER,
  4  interstate VARCHAR2(35),
  5  geom SDO_GEOMETRY
  6  );

Table created.

-- create table us_streets
SQL> CREATE TABLE us_streets
  2  (
  3  id NUMBER,
  4  street_name VARCHAR2(35),
  5  city VARCHAR2(32),
  6  state VARCHAR2(32),
  7  geom SDO_GEOMETRY
  8  );

Table created.
```

3.2 Metadata for Spatial Tables

- **Dictionary View for Spatial Metadata**
  The USER_SDO_GEOM_METADATA updatable view can be used to store metadata for spatial layers in Oracle Spatial.
  The USER_SDO_GEOM_METADATA View

```
SQL> DESCRIBE USER_SDO_GEOM_METADATA;

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TABLE_NAME                                NOT NULL VARCHAR2(32)
 COLUMN_NAME                               NOT NULL VARCHAR2(1024)
 DIMINFO                                            MDSYS.SDO_DIM_ARRAY
 SRID                                               NUMBER
```

  In this example:

- *'TABLE_NAME'* and *'COLUMN_NAM'* identify each spatial layer
- *'DIMINFO'* attribute stores information about the individual dimensions for the layer
- *'SRID'* attribute stores information about the manage system of the geometry data

DIMINFO Attribute

This attribute specifies the coordinate system where the data in the spatial layer is stored. The coordinate system could be either 'geodetic', 'projected', or 'local'.

For example, you can select the SRIDs in the case of geodetic coordinate systems to consult the CS_SRS1 table for possible values by selecting rows where the WKTEXT column2 starts with a prefix of 'GEOGCS'.

Selecting SRIDs of Geodetic Coordinate Systems

```
SQL> SELECT SRID
  2  FROM MDSYS.CS_SRS
  3  WHERE WKTEXT LIKE 'GEOGCS%';


......
      SRID
----------
  68056405
  68066405
  68086405
  68096405
  68136405
  68146405

844 rows selected.
```

For example, for the projected coordinate system from the MDSYS.CS_SRS table, search for rows where the WKTEXT column starts with 'PROJCS'.

Selecting SRIDs of Projected Coordinate Systems

```
SQL> SELECT SRID
  2  FROM MDSYS.CS_SRS
  3  WHERE WKTEXT LIKE 'PROJCS%';


......
      SRID
----------
      2767
      2768
      2769

3376 rows selected.
```

DIMINFO Attribute

The DIMINFO attribute in USER_SDO_GEOM_METADATA specifies information about each dimension of the specified layer. The DIMINFO attribute is of type MDSYS.SDO_DIM_ARRAY.

The USER_SDO_GEOM_METADATA View

```
SQL> DESCRIBE SDO_DIM_ARRAY;
 SDO_DIM_ARRAY VARRAY(4) OF MDSYS.SDO_DIM_ELEMENT
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SDO_DIMNAME                                        VARCHAR2(64)
 SDO_LB                                             NUMBER
 SDO_UB                                             NUMBER
 SDO_TOLERANCE                                      NUMBER
```

In this example:

- 'SDO_DIM_ARRAY' is a variable-length array (VARRAY) of type SDO_DIM_ELEMENT which each type stores information for a specific dimension

3.3 Populating Spatial Metadata for Your Application

- **Additional Information for Visualization and Network Analysis**

  Populate the tables in the sample application with metadata with USER_SDO_GEOM_METADATA view.

  The sample location is United States with the SRID 8307. This SRID is used in a majority of navigation systems that use Global Positioning Systems (GPS). This example use a tolerance value of 0.5 meters to insert a row in the USER_SDO_GEOM_METADATA view for the spatial layer corresponding to the location column of the customers table.

  Inserting Metadata for the Spatial Layer Corresponding to the location Column of the customer table:

```
SQL> INSERT INTO USER_SDO_GEOM_METADATA VALUES
  2  (
  3  'CUSTOMERS', -- TABLE_NAME
  4  'LOCATION', -- COLUMN_NAME
  5  SDO_DIM_ARRAY -- DIMINFO attribute for storing dimension bounds, tolerance
  6  (
  7   SDO_DIM_ELEMENT
  8   (
  9   'LONGITUDE', -- DIMENSION NAME for first dimension
 10   -180, -- SDO_LB for the dimension
 11   180, -- SDO_UB for the dimension
 12   0.5 -- Tolerance of 0.5 meters
 13   ),
```

```
14    SDO_DIM_ELEMENT
15    (
16    'LATITUDE', -- DIMENSION NAME for second dimension
17    -90, -- SDO_LB for the dimension
18    90, -- SDO_UB for the dimension
19    0.5 -- Tolerance of 0.5 meters
20    )
21   ),
22   8307 -- SRID value for specifying a geodetic coordinate system
23   );

1 row created.
```

In this example:

- the SRID of 8307 specifies that the data in the corresponding spatial layer are in a geodetic coordinate system
- the specific restrictions for identifying metadata for geodetic coordinate systems:
  - SDO_DIM_ARRAY
    - should correspond to the longitude dimension (the bounds should always be set to –180 and 180)
    - should correspond to the latitude dimension (the bounds should always be set to –90 and 90)
  - the tolerance for the dimensions must always be specified in “units”, meters

4.0 THE SDO_GEOMETRY DATA TYPE

In this section, the SDO_GEOMETRY data type is investigated and examined on the storage and modeling of location information. It is a powerful structure in Oracle that can be used to store point, line, polygon, surface, and solid geometries. It also can store homogenous and heterogeneous collections of such geometries. Particularly, the structure of SDO_GEOMETRY with different attributes and the values can store different types of geometric data, and the type of attributes can be described as:

- SDO_GTYPE attribute: specifies the type (shape)
- SDO_ELEM_INFO and SDO_ORDINATES attributes: specify the ordinate information and connectivity for the shape object
- SDO_POINT attribute: stores the location for two- or three-dimensional points

In addition, in an application, the SDO_GEOMETRY data type can be utilized to model locations of customers, delivery sites, and competitors as two-dimensional points or store the exact structure of buildings as three-dimensional solids in the location.

Creation of a geometry_examples table, which is used to store the SDO_GEOMETRY examples.

Creating a Table to Store All Geometry Examples:

```
SQL> CREATE TABLE geometry_examples
  2  (
  3  name VARCHAR2(100),
  4  description VARCHAR2(100),
  5  geom SDO_GEOMETRY
  6  );

Table created.
```

4.1 SDO_GEOMETRY Type, Attributes, and Values

The follow statement describes the structure of SDO_GEOMETRY data type in Oracle.

SDO_GEOMETRY Data Type in Oracle:

```
SQL> DESCRIBE SDO_GEOMETRY;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------------------
 SDO_GTYPE                                          NUMBER
 SDO_SRID                                           NUMBER
 SDO_POINT                                          MDSYS.SDO_POINT_TYPE
 SDO_ELEM_INFO                                      MDSYS. SDO_ELEM_INFO_ARRAY
 SDO_ORDINATES                                      MDSYS.SDO_ORDINATE_ARRAY
```

In this example:

- the name of attributes in SDO_GEOMETRY are listed with the data type which they can store

- **SDO_GTYPE Attribute**

SDO_GTYPE describes the type of geometric shape modeled in the object, which specifies the type of shape (point, line, polygon, collection, multipoint, multiline, or multipolygon) that the geometry actually represents.

The SDO_GTYPE attribute is a four-digit number structured as follows: D00T. The first digit 'D' (dimension of the geometry) and the last digits 'T' (shape/type of the geometry) take different values based on the dimensionality and shape of the geometry. The follow examples are used to examine the value type of SDO_GTYPE attribute.

Example of the SDO_GTYPE in the location Column of the customers Table, SELECT ct.location.sdo_gtype FROM customers ct WHERE id=1:

```
SQL> SELECT ct.location.sdo_gtype FROM customers ct WHERE id=1;

LOCATION.SDO_GTYPE
------------------
              2001
```

In this example:

- the value of T is 1, and SDO_GTYPE is 2001

Example of SDO_GTYPE in the geom Column of the us_interstates Table, SELECT i.geom.sdo_gtype FROM us_interstates i WHERE rownum=1:

```
SQL> SELECT i.geom.sdo_gtype FROM us_interstates i WHERE rownum=1;

GEOM.SDO_GTYPE
--------------
          2006
```

In this example:

- the value of T is 6, and SDO_GTYPE is 2006

Example of SDO_GTYPE in the location Column of the us_states Table, SELECT s.geom.sdo_gtype FROM us_states s WHERE state_abrv='NH':

```
SQL> SELECT s.geom.sdo_gtype FROM us_states s WHERE state_abrv='NH';

GEOM.SDO_GTYPE
--------------
         2003
```

In this example:

- he value of T is 3, and the geometry represents an area bounded by a closed line string (also referred to as ring) of edges

- **SDO_SRID Attribute**

SDO_SRID specifies the ID of the spatial reference system (coordinate system) in which the location/shape of the geometry is specified.
The follow statement is used to look up the SDO_SRIDs for the geodetic or projected coordinate systems in the MDSYS.CS_SRS table
MDSYS.CS_SRS Table:

```
SQL> DESCRIBE MDSYS.CS_SRS;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------------------------
 CS_NAME                                            VARCHAR2(80)
 SRID                                      NOT NULL NUMBER(38)
 AUTH_SRID                                          NUMBER(38)
 AUTH_NAME                                          VARCHAR2(256)
 WKTEXT                                             VARCHAR2(2046)
 CS_BOUNDS                                          MDSYS.SDO_GEOMETRY
```

In this example:

- The MDSYS.CS_SRS table has the six columns:
  - CS_NAME: specifies the name of the coordinate system
  - SRID: is short for spatial reference system ID
  - AUTH_SRID and AUTH_NAME: refer to the values assigned by the originator of this coordinate system
  - WKTEXT: is short for well-known text which provides a detailed description of the coordinate system
  - CS_BOUNDS: specifies a geometry where the coordinate system is valid and stores data beyond the bounds may lead to inaccurate results

Choosing an Appropriate Coordinate System

Examining the coordinate system description in the WKTEXT field can help with choosing the coordinate system.

This example identifies a projected coordinate system for southern Texas. (the ROWNUM=1 predicate displays only one out of three rows for southern Texas)

Selecting an SRID for the Southern Texas Region from the MDSYS.CS_SRS Table:

```
SQL> SELECT cs_name, srid, wktext
  2  FROM MDSYS.CS_SRS
  3  WHERE WKTEXT LIKE 'PROJCS%'
  4  AND CS_NAME LIKE '%Texas%Southern%'
  5  AND ROWNUM=1;

CS_NAME    SRID WKTEXT
--------------------------------------------------------------------------------
Texas 4205, Southern Zone (1927)
      41155
PROJCS["Texas 4205, Southern Zone (1927)", GEOGCS [ "NAD 27 (Continental US)", D
ATUM ["NAD 27 (Continental US)", SPHEROID ["Clarke 1866", 6378206.4, 294.9786982
]], PRIMEM [ "Greenwich", 0.000000 ], UNIT ["Decimal Degree", 0.0174532925199433
0]], PROJECTION ["Lambert Conformal Conic"], PARAMETER ["Standard_Parallel_1", 2
6.1666666667], PARAMETER ["Standard_Parallel_2", 27.8333333333], PARAMETER
["Central_Meridian", -98.500000], PARAMETER ["Latitude_Of_Origin", 25.6666666667], PA
RAMETER ["False_Easting",     2000000.0000], UNIT ["U.S. Foot", 0.3048006096012]]
```

In this example:

- the query returns a projected coordinate system for southern Texas
- the coordinate system: Lambert Conformal Conic
- projection technique: NAD 27 (continental United States)
- the corresponding SRID: 41155

The EPSG Coordinate System Model for 2-Dimensional and 3-Dimensional Data

EPSG is short for 'The European Petroleum Standards Group', which is a model used to provide more flexibility in transformations across coordinate systems. It supports a rich set of predefined, such as one-dimensional, two-dimensional and three-dimensional coordinate Systems.

- **Types of EPSG Coordinate Systems**

Below statement are used to determine the different types of supported two-dimensional and three-dimensional coordinate systems by querying the SDO_COORD_REF_SYS table.

Determining the Different Kinds of EPSG Coordinate Systems:

```
SQL> SELECT DISTINCT coord_ref_sys_kind FROM SDO_COORD_REF_SYS;

COORD_REF_SYS_KIND
------------------------
PROJECTED
GEOGRAPHIC2D
GEOCENTRIC
VERTICAL
ENGINEERING
COMPOUND
GEOGRAPHIC3D

7 rows selected.
```

In this example:

- Systems types to use:
  - 1D coordinate systems: Vertical
  - 2D coordinate systems: Geographic2D, Projected
  - 3D coordinate systems: Geographic3D, Geocentric, Compound
  - Local coordinate systems: Engineering

Searching for a Compound Coordinate System for the Texas Region:

```
SQL> SELECT srid, coord_ref_sys_name name,
  2  cmpd_horiz_srid hsrid, cmpd_vert_srid vsrid
  3  FROM sdo_coord_ref_sys
  4  WHERE coord_ref_sys_name like '%Texas%'
  5  AND coord_ref_sys_kind='COMPOUND';

      SRID      NAME                                       HSRID      VSRID
---------- ---------------------------------------- ---------- ----------
      7407      NAD27 / Texas North + NGVD29            32037          5702
```

In this example:

- the SRID of the compound coordinate system: 7407
- the SRID of horizontal coordinate system (pertains to the x,y or

longitude/latitude dimensions): 32037

- the SRID of vertical coordinate system (refers to the height from the surface of the earth): 5702

We can look up for the details of each particular pars of coordinate system searching by its SRID number.

Looking Up Details for Horizontal Coordinate System ID 32037:

```
SQL> SELECT cs_name, wktext FROM CS_SRS WHERE SRID=32037;

CS_NAME                  WKTEXT
--------------------------------------------------------------------------------
NAD27 / Texas NorthPROJCS["NAD27 / Texas North", GEOGCS [ "NAD27", DATUM ["North
American Datum 1927 (EPSG ID 6267)", SPHEROID ["Clarke 1866 (EPSG ID 7008)",
6378206.4, 294.97869821390582076161053712319517 5418], -3, 142, 183, 0, 0, 0, 0],
PRIMEM [ "Greenwich", 0.000000 ], UNIT ["Decimal Degree", 0.0174532925199 4328]],
PROJECTION ["Lambert Conformal Conic"], PARAMETER ["Latitude_Of_Origin", 34],
PARAMETER ["Central_Meridian", -101.5], PARAMETER ["Standard_Parallel_1", 34.65],
PARAMETER ["Standard_Parallel_2", 36.183333333333333333333333333333333333333],
PARAMETER ["False_Easting", 2000000], PARAMETER ["False_Northing", 0], UNIT ["U.S.
Foot", .304800609601219202438404876809753619 5072]
```

In this example:

- the return unit is "U.S. Foot"

Looking Up Details for Vertical Coordinate System ID 5702 in the CS_SRS Table:

```
SQL> SELECT cs_name, wktext FROM CS_SRS WHERE SRID=5702;

CS_NAME                                                  WKTEXT
-------------------------------------------------------- ----------------
National Geodetic Vertical Datum of 1929
```

In this example:

- the returns CS_NAME: “National Geodetic Vertical Datum of 1929”
- The query does not return any value for the wktext field

Looking Up Details for Vertical Coordinate System ID 5702 in Appropriate Tables:

```
SQL> SELECT cs.COORD_SYS_NAME
  2  FROM SDO_CRS_VERTICAL v, SDO_COORD_SYS cs
  3  WHERE v.SRID=5702 and cs.COORD_SYS_ID = v.COORD_SYS_ID;

COORD_SYS_NAME
--------------------------------------------------------------------------------
Gravity-related CS. Axis: height (H). Orientation: up.  UoM: ftUS.
```

In this example:

- the unit of measure (UoM) in the COORD_SYS_NAME that is ftUS, which is the same as U.S. Foot

- **Specifying a Preferred Transformation Path Between Coordinate Systems**

The EPSG model allows chained concatenations in transformations. The flow steps show an example of defining a transformation from use EPSG model:

Step 1: determine the EPSG-equivalent coordinate system (SRID) for SRID 41155

Finding an EPSG Equivalent for SRID 41155:

```
SQL> SELECT sdo_cs.find_proj_crs(41155, 'FALSE') epsg_srid FROM DUAL;

EPSG_SRID
--------------------------------------------------------------------------------
SDO_SRID_LIST(32041)
```

Step 2: identify the source geographic coordinate system on which this projected coordinate system is based on by querying the SDO_COORD_REF_SYS table

Finding the Source Geographic SRID and the projection_conversion ID for SRID 41155 (EPSG 32041):

```
SQL> SELECT projection_conv_id cid, source_geog_srid src_srid FROM
  2  SDO_COORD_REF_SYS
  3  WHERE srid=32041;

       CID    SRC_SRID
---------- ----------
     14205        4267
```

Step 3: define a preferred transformation from 41155 to 4269 by entering the transformation

Creating a Preferred Transformation Path Between 41155 (Projection-Based on NAD27) and 4269 (NAD83):

```
SQL> call sdo_cs.create_pref_concatenated_op(
  2  10000, -- any unique id of the operation,
  3  TFM_PLAN(
  4  SDO_TFM_CHAIN(
  5   41155, -- source srid
  6   14205, 4267, -- convid 14205 from srid 41155 (32041) to srid 4267
  7   1241, 4269 -- convid 1241 from 4267 to 4269
  8   ),
  9  NULL)
);
TFM_PLAN(
*
```

```
ERROR at line 3:
ORA-06553: PLS-306: wrong number or types of arguments in call to 'TFM_PLAN'
```

In this example:

- there is a problem when place a call to function TFM_PLAN, so the result can not be generated

- **SDO_POINT Attribute**

SDO_POINT specifies the location of a point geometry, such as the location of a customer.

SDO_POINT_TYPE Data Type:

```
SQL> DESCRIBE SDO_POINT_TYPE;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 X                                                  NUMBER
 Y                                                  NUMBER
 Z                                                  NUMBER
```

In this example:

- the structure of SDO_POINT types are described

An example of populate the SDO_GEOMETRY object in the geometry_examples table to represent point A (substitute (–79, 37) with actual coordinates)

Point Data in geometry_examples:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3    'POINT',
  4    '2-dimensional Point at coordinates (-79,37) with srid set to 8307',
  5    SDO_GEOMETRY
  6    (
  7      2001, -- SDO_GTYPE format: D00T. Set to 2001 for a 2-dimensional point
  8      8307, -- SDO_SRID (geodetic)
  9      SDO_POINT_TYPE
 10      (
 11        -79, -- ordinate value for Longitude
 12        37, -- ordinate value Latitude
 13        NULL -- no third dimension (only 2 dimensions)
 14      ),
 15      NULL,
 16      NULL
 17    )
 18  );

1 row created.
```

In this example:

- The geom column is an object of type SDO_GEOMETRY and is instantiated

showing below:

- SDO_GTYPE: format-D00T, where D is 2 and T is 1 for two-dimensional POINT
- SDO_SRID: 8307
- SDO_POINT: the x,y coordinates in SDO_POINT_TYPE are (−79, 37), and z coordinate is NULL
- SDO_ELEM_INFO: NULL
- SDO_ORDINATES: NULL

• This statement can be used to update the geom column value, or in spatial query operators and functions

An example of takes the WKT and an SRID as arguments to construct an SDO_GEOMETRY object in Oracle Spatial.

Constructing a Point Geometry Using Well-Known Text (SQL/MM):

```
SQL> SELECT SDO_GEOMETRY(' POINT(-79 37) ', 8307) geom FROM DUAL;

GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-79, 37, NULL), NULL, NULL)
```

4.2 Simple Two-Dimensional Geometry Examples

- **Point**

  SDO_ORDINATES array can be used to store the point coordinates of all elements of the geometry. The examples show how to store a 2-D point and a 4-D point in the SDO_ORDINATES.

  Storing the Point Coordinates in the SDO_ORDINATES Array Instead of SDO_POINT:

```
SQL> INSERT INTO geometry_examples VALUES
  2  (
  3    '2-D POINT stored in SDO_ORDINATES',
  4    '2-dimensional Point at coordinates (-79, 37) with srid set to 8307',
  5    SDO_GEOMETRY
  6    (
  7      2001, -- SDO_GTYPE format: D00T. Set to 2001 for as a 2-dimensional point
  8      8307, -- SDO_SRID
  9      NULL, -- SDO_POINT attribute set to NULL
 10        SDO_ELEM_INFO_ARRAY -- SDO_ELEM_INFO attribute (see Table 4-2 for
values)
 11      (
 12        1, -- Offset is 1
 13        1, -- Element-type is 1 for a point
 14        1 -- Interpretation specifies # of points. In this case 1.
 15      ),
 16      SDO_ORDINATE_ARRAY -- SDO_ORDINATES attribute
 17      (
 18        -79, -- Ordinate value for Longitude
 19        37 -- Ordinate value for Latitude
 20      )
 21    )
 22  );

1 row created.
```

  Storing a Four-Dimensional Point:

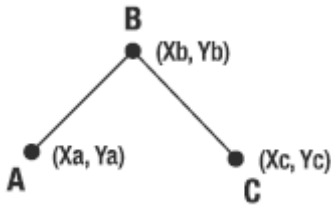
```
SQL> INSERT INTO geometry_examples VALUES
  2  (
  3    '4-D POINT',
  4    '4-dimensional Point at (Xa=>2, Ya=>2, Za=>2, La=>2) with srid set to NULL',
  5    SDO_GEOMETRY
  6    (
  7      4001, -- SDO_GTYPE: D00T. Set to 4001 as it is a 4-dimensional point
```

```
  8         NULL, -- SDO_SRID
  9         NULL, -- SDO_POINT_TYPE is null
 10         SDO_ELEM_INFO_ARRAY(1,1,1), -- Indicates a point element
 11         SDO_ORDINATE_ARRAY(2,2,2,2) -- Store the four ordinates here
 12       )
 13   );

1 row created.
```

In this example:

- Xa, Ya, Za, La are the ordinates of the four-dimensional point

- **Line String: Connected by Straight Lines**



Example of a line string connected by straight lines

An example of populate SDO_GEOMETRY object as a Two-Dimensional Line string.

Two-Dimensional Line string Example:

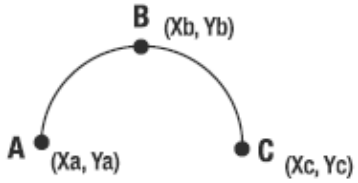
```
SQL> INSERT INTO geometry_examples VALUES
  2   (
  3     'LINE STRING',
  4     '2-D line string connecting A(Xa=>1,Ya=>1),B(Xb=>2, Yb=>2), C(Xc=>2,Yc=>1)',
  5     SDO_GEOMETRY
  6     (
  7       2002, -- SDO_GTYPE: D00T. Set to 2002 as it is a 2-dimensional line string
  8       32774, -- SDO_SRID
  9       NULL, -- SDO_POINT_TYPE is null
 10         SDO_ELEM_INFO_ARRAY -- SDO_ELEM_INFO attribute (see Table 4-2 for
values)
 11       (
 12         1, -- Offset is 1
 13         2, -- Element-type is 2 for a LINE STRING
 14         1 -- Interpretation is 1 if line string is connected by straight lines.
 15       ),
 16       SDO_ORDINATE_ARRAY -- SDO_ORDINATES attribute
 17       (
 18         1,1, -- Xa, Ya values
 19         2,2, -- Xb, Yb values
 20         2,1 -- Xc, Yc values
 21       )
```

```
 22      )
 23  );

1 row created.
```

In this example:

- the *SDO_ELEM_INFO* attribute is set to the triplet (1, 2, 1)
- the *SDO_ORDINATES* attribute is populated with the ordinates of each of the three vertices A, B, and C in the order they appear in the line string

- **Line String: Connected by Arcs**



Example of a line string connected by arcs

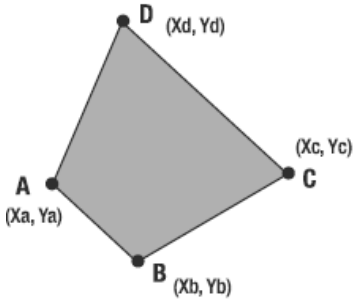
An example of stores a circular arc line string composed of three points.

Two-Dimensional Line String Connected by Arcs:

```
SQL> INSERT INTO geometry_examples VALUES
  2  (
  3     'ARCSTRING',
  4     '2-D arc connecting A(Xa=>1,Ya=>1),B(Xb=>2, Yb=>2), C(Xc=>2,Yc=>1)',
  5     SDO_GEOMETRY
  6     (
  7        2002, -- SDO_GTYPE: D00T. Set to 2002 as it is a 2-dimensional line string
  8        32774, -- SDO_SRID
  9        NULL, -- SDO_POINT_TYPE is null
 10          SDO_ELEM_INFO_ARRAY -- SDO_ELEM_INFO attribute (see Table 4-2 for
values)
 11        (
 12           1, -- Offset is 1
 13           2, -- Element-type is 2 for a LINE STRING
 14           2 -- Interpretation is 2 if line string is connected by ARCs.
 15        ),
 16        SDO_ORDINATE_ARRAY -- SDO_ORDINATES attribute
 17        (
 18           1,1, -- Xa, Ya values
 19           2,2, -- Xb, Yb values
 20           2,1 -- Xc, Yc values
 21        )
 22     )
 23  );
```

```
1 row created.
```

- **Polygon: Ring (Boundary) Connected by Straight Lines**

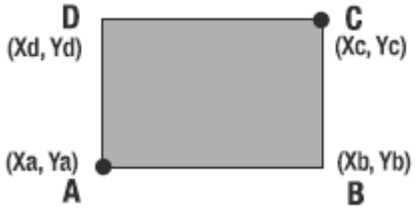


Example of a polygon boundary connected by lines

An example of storing a circular arc line string composed of three points.
Example of a Simple Polygon Connected by Lines:

```
SQL> INSERT INTO geometry_examples VALUES
  2  (
  3     'POLYGON',
  4     '2-D polygon connecting A(Xa, Ya), B(Xb, Yb), C(Xc, Yc), D(Xd, Yd)',
  5     SDO_GEOMETRY
  6     (
  7        2003,     -- SDO_GTYPE: D00T. Set to 2003 as it is a 2-dimensional polygon
  8        32774,    -- SDO_SRID
  9        NULL,     -- SDO_POINT_TYPE is null
 10          SDO_ELEM_INFO_ARRAY -- SDO_ELEM_INFO attribute (see Table 4-2 for
values)
 11        (
 12           1,          -- Offset is 1
 13           1003,     -- Element-type is 1003 for an outer POLYGON element
 14           1           -- Interpretation is 1 if boundary is connected by straight lines.
 15        ),
 16        SDO_ORDINATE_ARRAY -- SDO_ORDINATES attribute
 17        (
 18           1,1,       -- Xa, Ya values
 19           2,-1,     -- Xb, Yb values
 20           3,1,       -- Xc, Yc values
 21           2,2,       -- Xd, Yd values
 22           1,1        -- Xa, Ya values : Repeat first vertex to close the ring
 23        )
 24     )
 25  );

1 row created.
```

- **Rectangle Polygon**

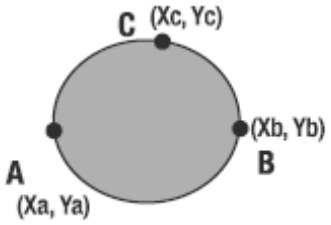


Example of a rectangular polygon

This is an example of inserting the polygon into the geometry_examples table.
Rectangular Polygon Example:

```
SQL> INSERT INTO geometry_examples VALUES
  2  (
  3     'RECTANGLE POLYGON',
  4     '2-D rectangle polygon with corner points A(Xa, Ya), C (Xc, Yc)',
  5     SDO_GEOMETRY
  6     (
  7        2003,        -- SDO_GTYPE: D00T. Set to 2003 as it is a 2-dimensional polygon
  8        32774,       -- SDO_SRID
  9        null,        -- SDO_POINT_TYPE is null
 10          SDO_ELEM_INFO_ARRAY -- SDO_ELEM_INFO attribute (see Table 4-2 for
values)
 11        (
 12           1,          -- Offset is 1
 13           1003,     -- Element-type is 1003 for (an outer) POLYGON
 14           3           -- Interpretation is 3 if polygon is a RECTANGLE
 15        ),
 16        SDO_ORDINATE_ARRAY -- SDO_ORDINATES attribute
 17        (
 18           1,1,       -- Xa, Ya values
 19           2,2        -- Xc, Yc values
 20        )
 21     )
 22  );

1 row created.
```

- **Circle Polygon**



Example of a circular polygon

Below is an example of inserting the circular polygon into the geometry_examples table.

Circular Polygon Example:

```
SQL> INSERT INTO geometry_examples VALUES
  2  (
  3     'CIRCLE POLYGON',
  4     '2-D circle polygon with 3 boundary points A(Xa,Ya), B(Xb,Yb), C(Xc,Yc)',
  5     SDO_GEOMETRY
  6     (
  7        2003,        -- SDO_GTYPE: D00T. Set to 2003 as it is a 2-dimensional polygon
  8        32774,       -- SDO_SRID
  9        NULL,         -- SDO_POINT_TYPE is null
 10          SDO_ELEM_INFO_ARRAY -- SDO_ELEM_INFO attribute (see Table 4-2 for
values)
 11        (
 12           1,          -- Offset is 1
 13           1003,     -- Element-type is 1003 for (an outer) POLYGON
 14           4           -- Interpretation is 4 if polygon is a CIRCLE
 15        ),
 16        SDO_ORDINATE_ARRAY -- SDO_ORDINATES attribute
 17        (
 18           1,1,       -- Xa, Ya values
 19           3,1,       -- Xb, Yb values
 20           2,2        -- Xc, Yc values
 21        )
 22     )
 23  );

1 row created.
```

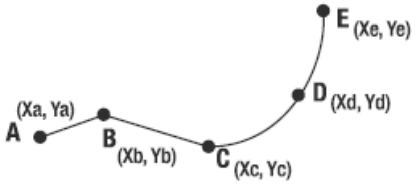
4.3 Complex Two-Dimensional Geometry Examples

In addition, there are three types in Two-Dimensional Geometries:

- a compound ling string (or a compound polygon)
- a voided polygon
- a collection

- **SDO_ELEM_INFO**
  SDO_ELEM_INFO specifies where in the SDO_ORDINATES array a new element starts, how it is connected (by straight lines or arcs), and whether it is a point, a line, or a polygon.

- **Voided Polygon Examples**
  Compound Line String



Example of a compound line string connected by lines and arcs

Compound Line String Example:

```
SQL> INSERT INTO geometry_examples VALUES
  2  (
  3    'COMPOUND LINE STRING',
  4    '2-D Compound Line String connecting A,B, C by a line and C, D, E by an arc',
  5    SDO_GEOMETRY
  6    (
  7      2002,          -- SDO_GTYPE: D00T. Set to 2002 as it is a 2-dimensional Line String
  8      32774,         -- SDO_SRID
  9      NULL,           -- SDO_POINT_TYPE is null
 10        SDO_ELEM_INFO_ARRAY -- SDO_ELEM_INFO attribute (see Table 4-2 for
values)
 11      (
 12        1,           -- Offset is 1
 13        4,           -- Element-type is 4 for Compound Line String
 14        2,           -- Interpretation is 2 representing number of subelements
 15        1, 2, 1,   -- Triplet for first subelement connected by line
 16        5, 2, 2     -- Triplet for second subelement connected by arc; offset is 5
 17      ),
 18      SDO_ORDINATE_ARRAY -- SDO_ORDINATES attribute
 19      (
 20        1,1,         -- Xa, Ya values for vertex A
```

```
 21            2,3,        -- Xb, Yb values for vertex B
 22            3,1,        -- Xc, Yc values for vertex C
 23            4,2,        -- Xd, Yd values for vertex D
 24            5,3          -- Xe, Ye values for vertex E
 25         )
 26      )
 27   );


1 row created.
```

- **Collection Examples**

Creating collection

SDO_UTIL.APPEND function is used for two non-overlapping geometries to returns an appended geometry.

For example, invoking APPEND using two polygons to get a result in multipolygon geometry.

Appending Two Geometries:

```
SQL> SELECT SDO_UTIL.APPEND
  2   (
  3      SDO_GEOMETRY
  4      (
  5         2003, 32774, null,
  6         SDO_ELEM_INFO_ARRAY(1,1003, 3),
  7         SDO_ORDINATE_ARRAY(1,1, 2,2)
  8      ),
  9      SDO_GEOMETRY
 10      (
 11         2003, 32774, NULL,
 12         SDO_ELEM_INFO_ARRAY(1, 1003, 3),
 13         SDO_ORDINATE_ARRAY(2,3, 4,5)
 14      )
 15   )
 16   FROM dual;


SDO_UTIL.APPEND(SDO_GEOMETRY(2003,32774,NULL,SDO_ELEM_INFO_ARRAY(1,10
03,3),SDO_O
--------------------------------------------------------------------------------
SDO_GEOMETRY(2007, 32774, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3, 5, 1003, 3),
SDO
_ORDINATE_ARRAY(1, 1, 2, 2, 2, 3, 4, 5))
```

For example, passing in a line and a polygon to get a heterogeneous collection (SDO_GTYPE =2007) geometry.

Creating a Heterogenous Collection Using SDO_UTIL.APPEND:

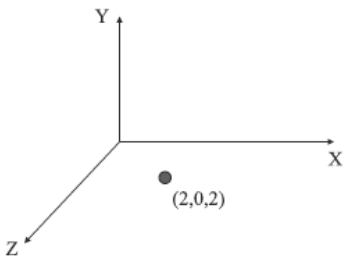
```
SQL> SELECT SDO_UTIL.APPEND
  2  (
  3     SDO_GEOMETRY
  4     (
  5        2003, 32774, null,
  6        SDO_ELEM_INFO_ARRAY(1,1003, 3),
  7        SDO_ORDINATE_ARRAY(1,1, 2,2)
  8     ),
  9     SDO_GEOMETRY
 10     (
 11        2002, 32774, NULL,
 12        SDO_ELEM_INFO_ARRAY(1, 2, 2),
 13        SDO_ORDINATE_ARRAY(2,3, 3,3,4,2)
 14     )
 15  )
 16  FROM dual;

SDO_UTIL.APPEND(SDO_GEOMETRY(2003,32774,NULL,SDO_ELEM_INFO_ARRAY(1,10
03,3),SDO_O
--------------------------------------------------------------------------------
SDO_GEOMETRY(2004, 32774, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3, 5, 2, 2),
SDO_OR
DINATE_ARRAY(1, 1, 2, 2, 2, 3, 3, 3, 4, 2))
```

4.4 Three-Dimensional Examples

SDO_GEOMETRY type can also be used to store different three-dimensional shapes that appear in a variety of applications. Below are some examples of examining SDO_GEOMETRY type in storing different type of three-dimensional shapes.

- **Three-Dimensional Points, Lines, and Polygons**

  Three-Dimensional Points

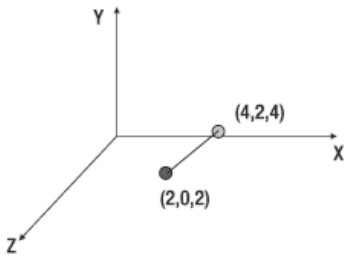


Three-dimensional point example

Three-Dimensional Point Geometry Example:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3    '3-D POINT',
  4    '3-dimensional Point at coordinates (2,0,2) ',
  5    SDO_GEOMETRY
  6    (
  7      3001,    -- SDO_GTYPE format: D00T. Set to 3001 for a 3-dimensional point
  8      NULL,    -- No coordinate system
  9      SDO_POINT_TYPE
 10      (
 11        2,     -- ordinate value for first dimension
 12        0,     -- ordinate value for second dimension
 13        2      -- ordinate value for third dimension
 14      ),
 15      NULL, -- SDO_ELEM_INFO is not needed as SDO_POINT field is populated
 16      NULL -- SDO_ORDINATES is not needed as SDO_POINT field is populated
 17    )
 18  );

1 row created.
```

Three-Dimensional Line String



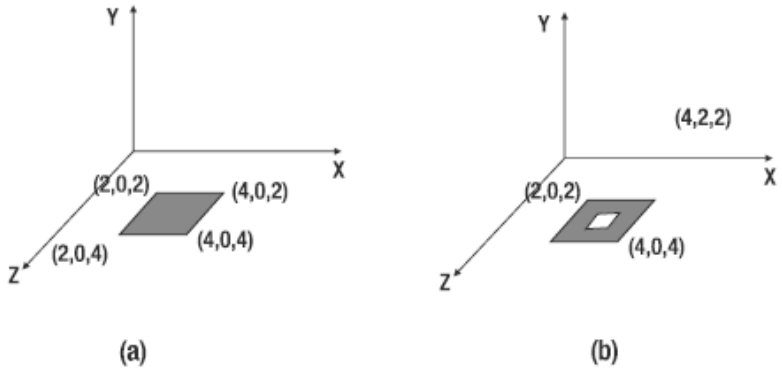
Three-dimensional line string example

Three-Dimensional Line String Geometry Example:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3    '3-D LineString',
  4    '3-dimensional LineString from coordinates (2,0,2) to (4,2,4) ',
  5    SDO_GEOMETRY
  6    (
  7      3002,    -- SDO_GTYPE format: D00T. Set to 3002 for a 3-dimensional line
  8      NULL,    -- No coordinate system
  9      NULL,    -- No data in SDO_POINT attribute
 10      SDO_ELEM_INFO_ARRAY(
 11        1,     -- Offset for ordinates in SDO_ORDINATE_ARRAY
```

```
12            2,      -- Line String typ
13            1       -- Connected by straight lines
14          ),
15          SDO_ORDINATE_ARRAY
16          (
17            2,      -- ordinate value for first dimension for first point
18            0,      -- ordinate value for second dimension for first point
19            2,      -- ordinate value for third dimension for first point
20            4,      -- ordinate value for first dimension for second point
21            2,      -- ordinate value for second dimension for second point
22            4       -- ordinate value for third dimension for second point
23          )
24      )
25  );

1 row created.
```

Three-Dimensional Polygon



Examples of a (a) three-dimensional polygon and (b) three-dimensional polygon with inner ring (hole)

SQL for Three-Dimensional Polygon (a) for the polygon element set to 1:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3    '3-D Polygon',
  4    '3-dimensional Polygon from coordinates (2,0,2) to (4,0, 4) ',
  5    SDO_GEOMETRY
  6    (
  7      3003,          -- SDO_GTYPE format: D00T. Set to 3003 for a 3-dimensional line
  8      NULL,          -- No coordinate system
  9      NULL,          -- No data in SDO_POINT attribute
 10      SDO_ELEM_INFO_ARRAY(
 11        1,           -- Offset for ordinates in SDO_ORDINATE_ARRAY
 12        3,           -- Polygon type
```

```
 13          1           -- Connected by straight lines
 14        ),
 15       SDO_ORDINATE_ARRAY
 16        (
 17          2, 0, 2,   -- coordinate values for first point
 18          2, 0, 4,   -- coordinate values for second point
 19          4, 0, 4,   -- coordinate values for third point
 20          4, 0, 2,   -- coordinate values for fourth point
 21          2, 0, 2     -- coordinate values for first point
 22        )
 23     )
 24  );

1 row created.
```

Three-Dimensional Rectangle Representation for Polygon (a) for the polygon element set to 3:

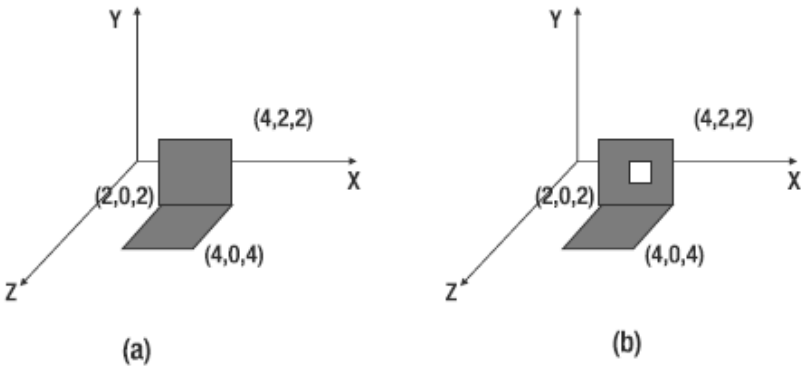
```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3     '3-D Rectangle Polygon',
  4     '3-dimensional Polygon from coordinates (2,0,2) to (4,0, 4) ',
  5     SDO_GEOMETRY
  6     (
  7        3003,          -- SDO_GTYPE format: D00T. Set to 3003 for a 3-dimensional polygon
  8        NULL,           -- No coordinate system
  9        NULL,           -- No data in SDO_POINT attribute
 10        SDO_ELEM_INFO_ARRAY(
 11           1,          -- Offset for ordinates of the Exterior Ring in SDO_ORDINATE_ARRAY
 12           1003,       -- ETYPE for Exterior ring
 13           3           -- MBR Connected by straight lines
 14        ),
 15        SDO_ORDINATE_ARRAY
 16        (
 17           2, 0, 2,   -- coordinates for min-corner of Exterior ring
 18           4, 0, 4     -- coordinates for max-corner of Exterior ring
 19        )
 20     )
 21  );

1 row created.
```

SQL for Polygon with Hole (b):

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
```

```
  3     '3-D Rectangle Polygon with Hole',
  4     '3-dimensional Polygon ',
  5     SDO_GEOMETRY
  6     (
  7       3003,              -- SDO_GTYPE format: D00T. Set to 3003 for a 3-dimensional
polygon
  8       NULL,               -- No coordinate system
  9       NULL,               -- No data in SDO_POINT attribute
 10       SDO_ELEM_INFO_ARRAY(
 11         1,                -- Offset for ordinates in SDO_ORDINATE_ARRAY
 12         1003,             -- Exterior ring etype
 13         3,                -- Rectangle
 14         7,                -- Offset for interior ring ordinates in SDO_ORDINATE_ARRAY
 15         2003,             -- ETYPE for Interior ring
 16         3                 -- Rectangle
 17       ),
 18       SDO_ORDINATE_ARRAY
 19       (
 20         2, 0, 2,          -- coordinates for min-corner of Exterior ring
 21         4, 0, 4,          -- coordinates for max-corner of Exterior ring
 22         3.5, 0, 3.5,      -- coordinates of max-corner of Interior ring
 23         3, 0, 3           -- coordinates of min-corner of Interior ring
 24       )
 25     )
 26   );

1 row created.
```

- **Composite Surfaces**



Examples of (a) a composite surface and (b) a closed composite surface

SQL for Composite Surface (a):

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
```

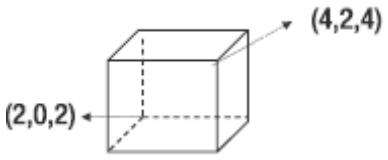
```
  3     '3-D Composite Surface',
  4     '3-dimensional Composite with 2 rectangle polygons ',
  5     SDO_GEOMETRY
  6     (
  7        3003,          -- SDO_GTYPE format: D00T. Set to 3003 for a 3-dimensional Surface
  8        NULL,           -- No coordinate system
  9        NULL,           -- No data in SDO_POINT attribute
 10        SDO_ELEM_INFO_ARRAY(
 11           1,          -- Offset of composite element
 12           1006,       -- Etype for composite surface element
 13           2,          -- Number of composing polygons
 14           1,          -- Offset for ordinates in SDO_ORDINATE_ARRAY
 15           1003,       -- Etype for Exterior (outer) ring of FIRST polygon
 16           3,          -- Polygon is an axis-aligned rectangle
 17           7,          -- Offset for second exterior polygon
 18           1003,       -- Etype for exterior Ring of SECOND polygon
 19           3           -- Polygon is an axis-aligned rectangle
 20        ),
 21        SDO_ORDINATE_ARRAY
 22        (
 23           2, 0,2,     -- min-corner for exterior ring of first polygon,
 24           4,2,2,      -- max-corner for exterior ring of first polygon
 25           2,0, 2,     -- min-corner for second element rectangle
 26           4,0,4       -- max-corner for second element rectangle
 27        )
 28     )
 29  );

1 row created.
```

SQL for Composite Surface (b):

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3     '3-D Composite Surface with hole polygons',
  4     '3-dimensional Composite with 2 rectangle polygons one of which has a hole ',
  5     SDO_GEOMETRY
  6     (
  7        3003,          -- SDO_GTYPE format: D00T. Set to 3003 for a 3-dimensional Surface
  8        NULL,           -- No coordinate system
  9        NULL,           -- No data in SDO_POINT attribute
 10        SDO_ELEM_INFO_ARRAY(
 11           1,          -- Offset of composite element
 12           1006,       -- Etype for composite surface element
 13           2,          -- Number of composing Polygons
```

```
14          1,          -- Offset for ordinates in SDO_ORDINATE_ARRAY
15          1003,       -- Etype for Exterior (outer) ring of FIRST polygon
16          3,          -- Polygon is an axis-aligned rectangle
17          7,          -- Offset for ordinates in SDO_ORDINATE_ARRAY
18          2003,       -- Etype for Interior (inner) ring of FIRST polygon
19          3,          -- Polygon is an axis-aligned rectangle
20          13,         -- Offset for second exterior polygon
21          1003,       -- Etype for exterior Ring of SECOND polygon
22          3           -- Polygon is an axis-aligned rectangle
23        ),
24        SDO_ORDINATE_ARRAY
25        (
26          2, 0, 2,        -- min-corner for exterior ring of first polygon,
27          4, 2, 2,        -- max-corner for exterior ring of first polygon
28          3, 1, 2,        -- min-corner for interior ring of first polygon
29          3.5, 1.5, 2,    -- max-corner for interior ring of first polygon
30          2,0, 2,          -- min-corner for second element rectangle
31          4,0,4             -- max-corner for second element rectangle
32        )
33      )
34    );

1 row created.
```

Example of the Surface of a Cube



A closed composite surface

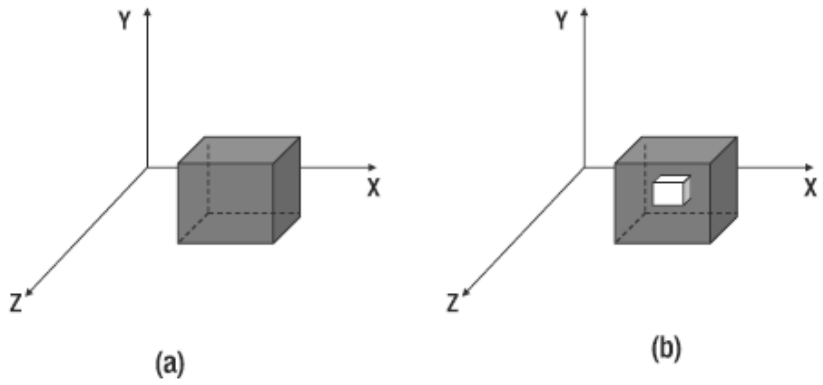
The composite surface consists of the six sides of the cube.

SQL for Composite Surface:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3    '3-D Composite Surface2',
  4    '3-dimensional Composite with 6 rectangle polygons ',
  5    SDO_GEOMETRY
  6    (
  7      3003,             -- SDO_GTYPE format: D00T. Set to 3003 for a 3-dimensional
Surface
  8      NULL,              -- No coordinate system
```

```
 9        NULL,                -- No data in SDO_POINT attribute
 10       SDO_ELEM_INFO_ARRAY
 11       (
 12         1,                 -- Offset of composite element
 13         1006,              -- Etype for composite surface element
 14         6,                 -- Number of composing polygons; element triplets for each
follow
 15         1, 1003,3,         -- Axis-aligned Rectangle element descriptor
 16         7, 1003,3,         -- Axis-aligned Rectangle element descriptor
 17         13,1003,3,         -- Axis-aligned Rectangle element descriptor
 18         19, 1003,3,        -- Axis-aligned Rectangle element descriptor
 19         25, 1003,3,        -- Axis-aligned Rectangle element descriptor
 20         31,1003,3          -- Axis-aligned Rectangle element descriptor
 21       ),
 22       SDO_ORDINATE_ARRAY
 23       (
 24         2,0,2, 4,2,2,    -- min-, max- corners for Back face,
 25         2,0,4, 4,2,4,      -- min-, max- corners for Front face,
 26         4,0,2, 4,2,4,      -- min-, max- corners for Right side face,
 27         2,0,2, 2,2,4,      -- min-, max- corners for Left side face,
 28         4,0,4, 2,0,2,      -- min-, max- corners for Bottom face,
 29         4,2,4, 2,2,2       -- min-, max- corners for Top face
 30       )
 31     )
 32   );

1 row created.
```

- **Simple Solid**



Examples of (a) simple solid and (b) a simple solid with a hole inside

SQL for Simple Solid (a):

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
```

```
  3     '3-D Simple Solid',
  4     '3-dimensional Solid with 6 rectangle polygons as its boundary ',
  5     SDO_GEOMETRY
  6     (
  7        3008,                  -- SDO_GTYPE format: D00T. Set to 3008 for a 3-dimensional
Solid
  8        NULL,                   -- No coordinate system
  9        NULL,                   -- No data in SDO_POINT attribute
 10        SDO_ELEM_INFO_ARRAY(
 11           1,                  -- Offset of a Simple solid element
 12           1007,               -- Etype for a Simple solid
 13           1,                  -- Indicates all surfaces are specified explicitly
 14           1,                  -- Offset of composite element
 15           1006,               -- Etype for composite surface element
 16           6,                  -- Number of composing elements
 17           -- element triplets for each element follow
 18           1, 1003,3,          -- Axis-aligned Rectangle element descriptor
 19           7, 1003,3,          -- Axis-aligned Rectangle element descriptor
 20           13, 1003,3,         -- Axis-aligned Rectangle element descriptor
 21           19, 1003,3,         -- Axis-aligned Rectangle element descriptor
 22           25, 1003,3,         -- Axis-aligned Rectangle element descriptor
 23           31, 1003,3          -- Axis-aligned Rectangle element descriptor
 24        ),
 25        SDO_ORDINATE_ARRAY
 26        (
 27           4,2,2, 2,0,2,     -- max-, min- corners for Back face (normal: -ve Z-axis)
 28           2,0,4, 4,2,4,     -- min-, max- corners for Front face (normal: +ve Z axis)
 29           4,0,2, 4,2,4,     -- min-, max- corners for Right face (normal: +ve X axis)
 30           2,2,4, 2,0,2,     -- min-, max- corners for Left face (normal: ?e X axis)
 31           4,0,4, 2,0,2,     -- max-, min- for Bottom face (normal: ?e Y axis)
 32           2,2,2, 4,2,4       -- min-, max- corners for Top face (normal: +ve Y axis)
 33        )
 34     )
 35  );

1 row created.
```

This is an example of using a box element and only two corners for a simple solid element.

SQL for Simple Solid (a):

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3     '3-D Simple Solid as a Solid Box',
  4     '3-dimensional Solid with just the 2 corner vertices ',
```

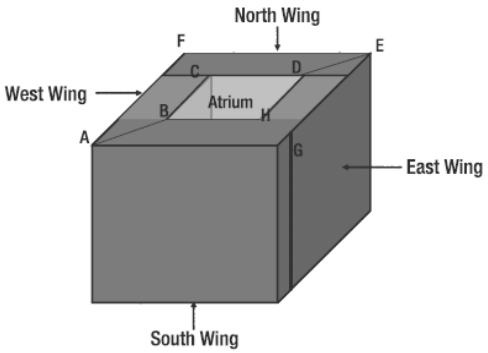
```
  5     SDO_GEOMETRY
  6     (
  7        3008,               -- SDO_GTYPE format: D00T. Set to 3008 for a 3-dimensional
Solid
  8        NULL,                -- No coordinate system
  9        NULL,                -- No data in SDO_POINT attribute
 10        SDO_ELEM_INFO_ARRAY(
 11           1,                -- Offset of a Simple solid element
 12           1007,             -- Etype for a Simple solid
 13           3                 -- Solid Box type: only two corner vertices are specified
 14        ),
 15        SDO_ORDINATE_ARRAY
 16        (
 17           2,0,2, 4,2,4   -- min-corner and max-corner for the solid
 18        )
 19     )
 20  );

1 row created.
```

Simple Solid with an Inner Hole (b):

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3     '3-D Simple Solid with inner hole',
  4     '3-dimensional Solid with 6 rectangle polygons as its boundary ',
  5     SDO_GEOMETRY
  6     (
  7        3008,             -- SDO_GTYPE format: D00T. Set to 3008 for a 3-dimensional Solid
  8        NULL,              -- No coordinate system
  9        NULL,              -- No data in SDO_POINT attribute
 10        SDO_ELEM_INFO_ARRAY(
 11           1,             -- Offset of a Simple solid element
 12           1007,           -- Etype for a Simple solid
 13           1,             -- Indicates all surfaces are specified explicitly
 14           1,             -- Offset of composite element
 15           1006,           -- Etype for composite surface element
 16           6,             -- # of composing elements; element triplets for each element follow
 17           1, 1003,3,    -- Axis-aligned Rectangle element descriptor
 18           7, 1003,3,    -- Axis-aligned Rectangle element descriptor
 19           13,1003,3,    -- Axis-aligned Rectangle element descriptor
 20           19, 1003,3, -- Axis-aligned Rectangle element descriptor
 21           25, 1003,3, -- Axis-aligned Rectangle element descriptor
 22           31, 1003,3, -- Axis-aligned Rectangle element descriptor
 23           37, 2006,6, -- Inner composite surface
```

```
24            37, 2003,3, -- Axis-aligned Rectangle element ; note etype is 2003
25            43, 2003,3, -- Axis-aligned Rectangle element descriptor
26            49, 2003,3, -- Axis-aligned Rectangle element descriptor
27            55, 2003,3, -- Axis-aligned Rectangle element descriptor
28            61, 2003,3, -- Axis-aligned Rectangle element descriptor
29            67, 2003,3   -- Axis-aligned Rectangle element descriptor
30          ),
31          SDO_ORDINATE_ARRAY
32          (
33            --- All polygons oriented such that normal point outward from solid
34            ---
35            ------- Ordinates for the rectangles of the outer composite surface
36            4,2,2, 2,0,2, -- Back face
37            2,0,4, 4,2,4, -- Front face
38            4,0,2, 4,2,4, -- Right face
39            2,2,4, 2,0,2, -- Left face
40            4,0,4, 2,0,2, -- Bottom face
41            2,2,2, 4,2,4, -- Top face
42            ---
43            ------- Ordinates for the rectangles of inner/hole composite surface
44            -------- representing the atrium
45            2.5, 0.5, 2.5, 3.5, 1.5, 2.5, -- Back face
46            3.5, 1.5, 3.5, 2.5, 0.5, 3.5, -- Front face
47            3.5, 1.5, 3.5, 3.5, 0.5, 2.5, -- Right face
48            2.5, 0.5, 2.5, 2.5, 1.5, 3.5, -- Left face
49            2.5, 0.5, 2.5, 3.5, 0.5, 3.5, -- Bottom face
50            3.5, 1.5, 3.5, 2.5, 1.5, 2.5    -- Top face
51          )
52      )
53  );


1 row created.
```

Solid with a Hole



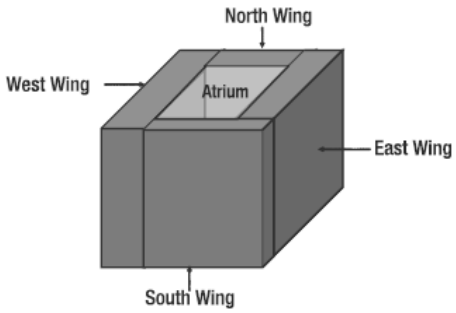
A solid with the hole where the hole cannot be modeled as an inner surface

Simple Solid with an Inner Hole That Touches Both Top and Bottom Faces:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3    '3-D Simple Solid with inner hole touching top/bottom faces',
  4    '3-dimensional Solid with 8 rectangle polygons as its boundary ',
  5    SDO_GEOMETRY
  6    (
  7      3008,             -- SDO_GTYPE format: D00T. Set to 3008 for a 3-dimensional Solid
  8      NULL,              -- No coordinate system
  9      NULL,              -- No data in SDO_POINT attribute
 10      SDO_ELEM_INFO_ARRAY(
 11        1,              -- Offset of a Simple solid element
 12        1007,           -- Etype for a Simple solid
 13        1,              -- Indicates all surfaces are specified explicitly
 14        1,              -- Offset of composite element
 15        1006,           -- Etype for composite surface element
 16        8,              -- # of composing elements; element triplets for each element follow
 17        1, 1003,3,    -- Axis-aligned Rectangle element descriptor for left face
 18        7, 1003,3,    -- Axis-aligned Rectangle element descriptor for right face
 19        13,1003,3,    -- Axis-aligned Rectangle element descriptor for back face
 20        19,1003,3,    -- Axis-aligned Rectangle element descriptor for front face
 21        25,1003,1,    -- Element descriptor for ABCDEFA on Top Face
 22        46,1003,1,    -- Element descriptor for AGEDHBA on Top Face
 23        67,1003,1,    -- Element descriptor for equivalent ABCDEFA on Bottom Face
 24        88,1003,1      -- Element descriptor for equivalent AGEDHBA on Bottom Face
 25      ),
 26      SDO_ORDINATE_ARRAY
 27      (
 28        -- Outer side walls
 29        4,2,2, 2,0,2, -- Back face
 30        2,0,4, 4,2,4, -- Front face
 31        4,0,2, 4,2,4, -- Right side face
```

```
32            2,2,4, 2,0,2, -- Left side face
33            --
34            -- Inner side walls
35            2.5,0,2.5, 3.5,2,2.5, -- Back Face
36            3.5,2,3.5, 2.5,0,3.5, -- Front Face
37            2.5,0,2.5, 2.5,2,3.5, -- Left Face
38            3.5,2,3.5, 3.5,0,3.5, -- Right Face
39            --
40            -- Coordinates for vertices A,B,C,D,E,F,A on top face
41            2,2,4, 2.5,2,3.5, 2.5,2,2.5, 3.5,2,2.5, 4,2,2, 2,2,2, 2,2,4,
42            -- Coordinates for vertices A,G,E,D,H,B,A on top face
43            2,2,4, 4,2,4, 4,2,2, 3.5,2,2.5, 3.5,2,3.5, 2.5,2,3.5, 2,2,4,
44            -- Coordinates for polygon equivalent to ABCDEFA on bottom face
45            2,0,4, 2,0,2, 4,0,2, 3.5,0,2.5, 2.5,0,2.5, 2.5,0,3.5, 2,0,4,
46            -- Coordinates for polygon equivalent to AGEDHBA on bottom face
47            2,0,4, 2.5,0,3.5, 3.5,0,3.5, 3.5,2,2.5, 4,0,2, 4,0,4, 2,0,4
48         )
49      )
50   );

1 row created.
```

- **Composite Solid**



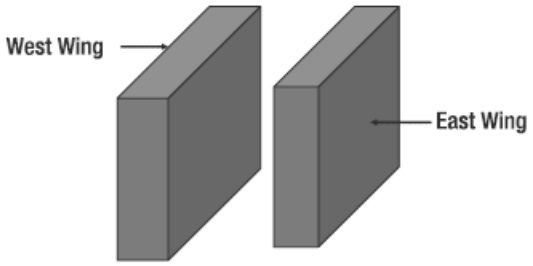
A composite solid

SQL for Composite Solid:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2   (
  3      '3-D Composite Solid of 4 simple solids',
  4      '3-dimensional composite solid ',
  5      SDO_GEOMETRY
  6      (
  7         3008,        -- SDO_GTYPE format: D00T. Set to 3008 for a 3-dimensional Solid
  8         NULL,         -- No coordinate system
  9         NULL,         -- No data in SDO_POINT attribute
 10         SDO_ELEM_INFO_ARRAY(
```

```
11          1,          -- Offset of the composite solid element
12          1008,       -- Etype for a composite solid
13          4,          -- Number of cimple solids making up the composite.
14          -- The simple solid descriptors next.
15          1, 1007, 3,     -- Simple solid as a solid Box
16          7, 1007, 3,     -- Simple solid as a solid box
17          13, 1007, 3,    -- Simple solid as a solid box
18          19, 1007, 3     -- Simple solid as a solid box
19        ),
20        SDO_ORDINATE_ARRAY
21        (
22          -- min-corner and max-corner for the West wing
23          2,0,2, 2.5,2,4,
24          -- min-corner and max-corner for the East wing,
25          3.5, 0,2, 4,2,3.5,
26          -- min-corner and max-corner for the North wing,
27          2.5,0,2, 3.5,2,2.5,
28          -- min-corner and max-corner for the South wing,
29          2.5,0,3.5, 4,2,4
30        )
31      )
32    );

1 row created.
```

- **Collections**



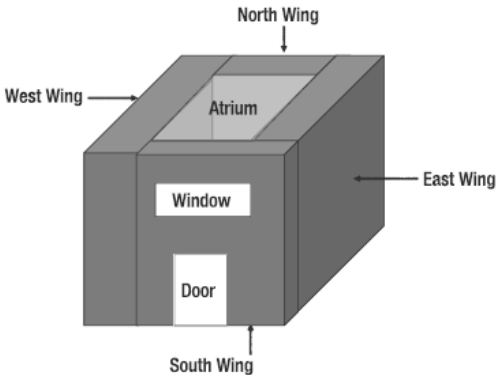
Different disjoint parts of a building as a multisolid

Example SQL for Multisolid:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3    '3-D Multi Solid',
  4    '3-dimensional Multisolid with 2 solid boxes ',
  5    SDO_GEOMETRY
  6    (
  7      3009,          -- SDO_GTYPE format: D00T. Set to 3009 for a 3-dimensional
MultiSolid
```

```
 8        NULL,              -- No coordinate system
 9        NULL,              -- No data in SDO_POINT attribute
10        SDO_ELEM_INFO_ARRAY(
11           1,              -- Offset of a simple solid element
12           1007,           -- Etype for a simple solid
13           3,              -- Solid box type: only two corner vertices are specified
14           7, 1007, 3   -- Solid Box for second solid
15        ),
16        SDO_ORDINATE_ARRAY
17        (
18           -- min-corner and max-corner for first solid
19           0,0,0, 4,4,4,
20           -- min-corner and max-corner for second solid.
21           5,0,0, 9,4,4
22        )
23     )
24  );

1 row created.
```

Example of an Entire Building



A model of entire building (with windows, doors) as a (heterogeneous) collection

Example SQL for Buildings Modeled As a Collection:

```
SQL> INSERT INTO geometry_examples (name, description, geom) VALUES
  2  (
  3     '3-D Building as a Collection',
  4     '3-dimensional collection as combination of a composite solid and 2 surfaces',
  5     SDO_GEOMETRY
  6     (
  7        3004,              -- SDO_GTYPE format: D00T. Set to 3004 for a 3-dimensional
Collection
  8        NULL,              -- No coordinate system
  9        NULL,              -- No data in SDO_POINT attribute
```

```
10          SDO_ELEM_INFO_ARRAY(
11            1, 1008, 4,     -- Descriptor for a composite solid of 4 simple solids
12            1, 1007, 3,     -- Simple solid as a solid Box
13            7, 1007, 3,     -- Simple solid as a solid box
14            13, 1007, 3,    -- Simple solid as a solid box
15            19, 1007, 3,    -- Simple solid as a solid box,
16            25, 1003, 3,    -- Descriptor for Door as a polygon
17            31, 1003, 3     -- Descriptor for Window as a polygon
18          ),
19          SDO_ORDINATE_ARRAY
20          (
21            -- min-corner and max-corner for the West wing
22            2,0,2, 2.5,2,4,
23            -- min-corner and max-corner for the East wing,
24            3.5, 0,2, 4,2,3.5,
25            -- min-corner and max-corner for the North wing,
26            2.5,0,2, 3.5,2,2.5,
27            -- min-corner and max-corner for the South wing,
28            2.5,0,3.5, 4,2,4,
29            -- min-corner and max-corner for the door,
30            2.75, 0, 4, 3.25, 1, 4,
31            -- min-corner and max-corner for the window,
32            2.5, 2, 4, 3.5, 3, 4
33          )
34        )
35    );

1 row created.
```

5.0 LOADING, TRANSPORTING, AND VALIDATING

This section manipulates different ways to populate Oracle tables that contain SDO_GEOMETRY columns. These tables could be part of an e-business application or in CAD/CAM, GIS, GPS, wireless, or telemetric applications. Moreover, the utility of functions/procedures are tested with examples for importing and transferring data between Oracle databases. Finally, examples are used to validate the loaded SDO_GEOMETRY objects and correct invalid objects.

5.1 Inserting Data into an SDO_GEOMETRY Column

Create a table to model the sales regions of a business franchise.

Creating the sales_regions Table:

```
SQL> CREATE TABLE sales_regions
  2  (
  3     id NUMBER,
  4     geom SDO_GEOMETRY
  5  );

Table created.
```

Insert polygons representing sales regions into the geom column of this table.

Inserting a Polygon Geometry into the sales_regions Table:

```
SQL> INSERT INTO sales_regions VALUES
  2  (
  3     10000,            -- SALES_REGIONS ID
  4     SDO_GEOMETRY   -- use the SDO_GEOMETRY constructor
  5     (
  6        2003,          -- A two-dimensional Polygon
  7        8307,          -- SRID is GEODETIC
  8        NULL,           -- SDO_POINT_TYPE is null as it is not a point
  9        SDO_ELEM_INFO_ARRAY (1, 1003, 1), -- A polygon with just one ring
 10        SDO_ORDINATE_ARRAY -- SDO_ORDINATES field
 11        (
 12           -77.04487, 38.9043742, -- coordinates of first vertex
 13           -77.046645, 38.9040983, -- other vertices
 14           -77.04815, 38.9033127, -77.049155, 38.9021368,
 15           -77.049508, 38.9007499, -77.049155, 38.899363, -77.048149, 38.8981873,
 16           -77.046645, 38.8974017, -77.04487, 38.8971258, -77.043095, 38.8974017,
 17           -77.041591, 38.8981873, -77.040585, 38.899363, -77.040232, 38.9007499,
 18           -77.040585, 38.9021368, -77.04159, 38.9033127, -77.043095, 38.9040983,
```

```
 19              -77.04487, 38.9043742 -- coordinates of last vertex same as first vertex
 20          )
 21       )
 22    );

1 row created.
```

5.2 Loading and Converting Spatial Data

- **Converting Between SDO_GEOMETRY and WKT/WKB**
  Below is an example of converting an SDO_GEOMETRY object to WKT format.
  Converting from an SDO_GEOMETRY to WKT Format:

```
SQL> SELECT a.location.GET_WKT() wkt FROM customers a WHERE id=1;

WKT
--------------------------------------------------------------------------------
POINT (-77.06 38.94)

SQL> SELECT SDO_UTIL.TO_WKTGEOMETRY(a.location) wkt FROM customers a WHERE
id=1;

WKT
--------------------------------------------------------------------------------
POINT (-77.06 38.94)
```

  Below is an example of using the SDO_UTIL.TO_WKTGEOMETRY function to get the same result.
  Using TO_WKTGEOMETRY to Convert from an SDO_GEOMETRY to WKT Format:

```
SQL> SELECT SDO_UTIL.TO_WKTGEOMETRY(a.location) wkt FROM customers a WHERE
id=1;

WKT
--------------------------------------------------------------------------------
POINT (-77.06 38.94)

SQL> SELECT SDO_UTIL.TO_WKTGEOMETRY(a.location) wkt FROM customers a WHERE
id=1;

WKT
--------------------------------------------------------------------------------
POINT (-77.06 38.94)
```

- **Converting SDO_GEOMETRY Data in GML**

  GML is short for Geographic Markup Language, which is an XML-based encoding standard for spatial information. In addition, various functions in the SDO_UTIL package can be used to convert SDO_GEOMETRY data to/from GML format.

  Convert to GML

  Below is an example of converting a customer location into a GML document fragment.

  Converting an SDO_GEOMETRY to a GML Document:

```
SQL> SELECT TO_CHAR(SDO_UTIL.TO_GMLGEOMETRY(location)) gml_location
  2  FROM customers
  3  WHERE id=1;

GML_LOCATION
--------------------------------------------------------------------------------
<gml:Point srsName="SDO:8307" xmlns:gml="http://www.opengis.net/gml"><gml:coordi
nates decimal="." cs="," ts=" ">-77.06,38.94 </gml:coordinates></gml:Point>
```

  In this example:

  - between <*gml*> and </*gml*> tags: the geometry information
  - GML type: POINT
  - between the <*gml:coordinates*> and </*gml:coordinates*> tags: coordinates

  Below is an example of using TO_GMLGEOMETRY311 function in the SDO_UTIL package to convert an axis-aligned solid box into GML311.

  Converting a Three-Dimensional Solid SDO_GEOMETRY to GML311:

```
SQL> SELECT TO_CHAR(
  2     SDO_UTIL.TO_GML311GEOMETRY (
  3        SDO_GEOMETRY
  4        (
  5           3008, -- SDO_GTYPE format: D00T. Set to 3008 for a 3-dimensional Solid
  6           NULL, -- No coordinate system
  7           NULL, --- No data in SDO_POINT attribute
  8           SDO_ELEM_INFO_ARRAY (
  9              1, -- Offset of a Simple solid element
 10              1007, --- Etype for a Simple solid
 11              3 -- Indicates an axis-aligned box
 12           ),
 13           SDO_ORDINATE_ARRAY (
 14              0,0,0, --min-corners for box
 15              4,4,4 --min-corners for box
 16           )
 17        )
 18     )
 19  ) AS GML_GEOMETRY FROM DUAL;
```

```
GML_GEOMETRY
--------------------------------------------------------------------------------
<gml:Solid srsName="SDO:" xmlns:gml="http://www.opengis.net/gml"><gml:exterior><
gml:CompositeSurface><gml:surfaceMember><gml:Polygon><gml:exterior><gml:LinearRi
ng><gml:posList srsDimension="3">0.0 0.0 0.0 0.0 4.0 0.0 4.0 4.0 0.0 4.0 0.0 0.0
  0.0 0.0 0.0 </gml:posList></gml:LinearRing></gml:exterior></gml:Polygon></gml:s
urfaceMember><gml:surfaceMember><gml:Polygon><gml:exterior><gml:LinearRing><gml:
posList srsDimension="3">4.0 4.0 4.0 0.0 4.0 4.0 0.0 0.0 4.0 4.0 0.0 4.0 4.0 4.0
  4.0 </gml:posList></gml:LinearRing></gml:exterior></gml:Polygon></gml:surfaceMe
mber><gml:surfaceMember><gml:Polygon><gml:exterior><gml:LinearRing><gml:posList
srsDimension="3">0.0 0.0 0.0 4.0 0.0 0.0 4.0 0.0 4.0 0.0 0.0 4.0 0.0 0.0 0.0 </g
ml:posList></gml:LinearRing></gml:exterior></gml:Polygon></gml:surfaceMember><gm
l:surfaceMember><gml:Polygon><gml:exterior><gml:LinearRing><gml:posList srsDimen

GML_GEOMETRY
--------------------------------------------------------------------------------
sion="3">0.0 0.0 0.0 0.0 0.0 4.0 0.0 4.0 4.0 0.0 4.0 0.0 0.0 0.0 0.0 </gml:posLi
st></gml:LinearRing></gml:exterior></gml:Polygon></gml:surfaceMember><gml:surfac
eMember><gml:Polygon><gml:exterior><gml:LinearRing><gml:posList srsDimension="3"
>4.0 4.0 4.0 4.0 4.0 0.0 0.0 4.0 0.0 0.0 4.0 4.0 4.0 4.0 4.0 </gml:posList></gml
:LinearRing></gml:exterior></gml:Polygon></gml:surfaceMember><gml:surfaceMember>
<gml:Polygon><gml:exterior><gml:LinearRing><gml:posList srsDimension="3">4.0 4.0
  4.0 4.0 0.0 4.0 4.0 0.0 0.0 4.0 4.0 0.0 4.0 4.0 4.0 </gml:posList></gml:LinearR
ing></gml:exterior></gml:Polygon></gml:surfaceMember></gml:CompositeSurface></gm
l:exterior></gml:Solid>
```

Below is an example of using the XMLFOREST function to encode multiple geometries in a GML document.

Converting Multiple Geometries to a GML Document Fragment:

```
SQL> SELECT xmlelement(
  2      "State",
  3      xmlattributes('http://www.opengis.net/gml' as "xmlns:gml"),
  4      xmlforest(state as "Name", totpop as "Population",
  5      xmltype(sdo_util.to_gmlgeometry(geom)) as
  6      "gml:geometryProperty"))
  7    AS theXMLElements
  8    FROM US_STATES
  9    WHERE state_abrv in ('DE', 'UT');

THEXMLELEMENTS
--------------------------------------------------------------------------------
<State xmlns:gml="http://www.opengis.net/gml"><Name>Delaware</Name><Population>6
<State xmlns:gml="http://www.opengis.net/gml"><Name>Utah</Name><Population>17228
```

Converting GML to SDO_GEOMETRY

When converting GML geometry fragments to SDO_GEOMETRY there are tow function in the SDO_UTIL package can be used, which are FROM_GMLGEOMETRY and FROM_GML311GEOMETRY.

An example of converting the GML_GEOMETRY output back to an SDO_GEOMETRY. (GML311 to Three-Dimensional Solid SDO_GEOMETRY)

Converting a GML Solid Geometry into an SDO_GEOMETRY:

```
SQL> SELECT SDO_UTIL.FROM_GML311GEOMETRY(
  2  TO_CLOB(
  3  '<gml:Solid srsName="SDO:" xmlns:gml="http://www.opengis.net/gml">
  4  <gml:exterior>
  5  <gml:CompositeSurface>
  6  <gml:surfaceMember>
  7  <gml:Polygon>
  8  <gml:exterior>
  9  <gml:LinearRing>
 10  <gml:posList srsDimension="3">
 11  0.0 0.0 0.0 0.0 4.0 0.0 4.0 4.0 0.0 4.0 0.0 0.0 0.0 0.0 0.0
 12  </gml:posList>
 13  </gml:LinearRing>
 14  </gml:exterior>
 15  </gml:Polygon>
 16  </gml:surfaceMember>
 17  <gml:surfaceMember>
 18  <gml:Polygon>
 19  <gml:exterior>
 20  <gml:LinearRing>
 21  <gml:posList srsDimension="3">
 22  4.0 4.0 4.0 0.0 4.0 4.0 0.0 0.0 4.0 4.0 0.0 4.0 4.0 4.0 4.0
 23  </gml:posList>
 24  </gml:LinearRing>
 25  </gml:exterior>
 26  </gml:Polygon>
 27  </gml:surfaceMember>
 28  <gml:surfaceMember>
 29  <gml:Polygon>
 30  <gml:exterior>
 31  <gml:LinearRing>
 32  <gml:posList srsDimension="3">
 33  0.0 0.0 0.0 4.0 0.0 0.0 4.0 0.0 4.0 0.0 0.0 4.0 0.0 0.0 0.0
 34  </gml:posList>
 35  </gml:LinearRing>
 36  </gml:exterior>
 37  </gml:Polygon>
```

```
 38  </gml:surfaceMember>
 39  <gml:surfaceMember>
 40  <gml:Polygon>
 41  <gml:exterior>
 42  <gml:LinearRing>
 43  <gml:posList srsDimension="3">
 44  0.0 0.0 0.0 0.0 0.0 4.0 0.0 4.0 4.0 0.0 4.0 0.0 0.0 0.0 0.0
 45  </gml:posList>
 46  </gml:LinearRing>
 47  </gml:exterior>
 48  </gml:Polygon>
 49  </gml:surfaceMember>
 50  <gml:surfaceMember>
 51  <gml:Polygon>
 52  <gml:exterior>
 53  <gml:LinearRing>
 54  <gml:posList srsDimension="3">
 55  4.0 4.0 4.0 4.0 4.0 0.0 0.0 4.0 0.0 0.0 4.0 4.0 4.0 4.0 4.0
 56  </gml:posList>
 57  </gml:LinearRing>
 58  </gml:exterior>
 59  </gml:Polygon>
 60  </gml:surfaceMember>
 61  <gml:surfaceMember>
 62  <gml:Polygon>
 63  <gml:exterior>
 64  <gml:LinearRing>
 65  <gml:posList srsDimension="3">
 66  4.0 4.0 4.0 4.0 0.0 4.0 4.0 0.0 0.0 4.0 4.0 0.0 4.0 4.0 4.0
 67  </gml:posList>
 68  </gml:LinearRing>
 69  </gml:exterior>
 70  </gml:Polygon>
 71  </gml:surfaceMember>
 72  </gml:CompositeSurface>
 73  </gml:exterior>
 74  </gml:Solid>'
 75  )) GEOM FROM DUAL;

GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(3008, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1007, 1, 1, 1006, 6, 1,
10
```

```
03, 1, 16, 1003, 1, 31, 1003, 1, 46, 1003, 1, 61, 1003, 1, 76, 1003, 1), SDO_ORD
INATE_ARRAY(0, 0, 0, 0, 4, 0, 4, 4, 0, 4, 0, 0, 0, 0, 0, 4, 4, 4, 0, 4, 4, 0, 0,
 4, 4, 0, 4, 4, 4, 4, 0, 0, 0, 4, 0, 0, 4, 0, 4, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0
, 4, 0, 4, 4, 0, 4, 0, 0, 0, 0, 4, 4, 4, 4, 4, 0, 0, 4, 0, 0, 4, 4, 4, 4, 4, 4,
4, 4, 4, 0, 4, 4, 0, 0, 4, 4, 0, 4, 4, 4))
```

5.3 Extruding a 2-Dimensional Geometry to 3-Dimensions

In an application which store the two-dimensional footprints of buildings and other three-dimensional objects, the function EXTRUDE in the SDO_UTIL package can be used to upright a building on the two-dimensional footprint with specification of the ground height and the top height for each vertex of the two-dimensional geometry
An example of extruding the two-dimensional polygon by specifying the ground height as −1 and top height as 1.

Extruding a Polygon to a Three-Dimensional Solid:

```
SQL> SELECT SDO_UTIL.EXTRUDE(
  2     SDO_GEOMETRY -- first argument to validate is geometry
  3     (
  4        2003, -- 2-D Polygon
  5        NULL,
  6        NULL,
  7        SDO_ELEM_INFO_ARRAY(
  8           1, 1003, 1                    -- A polygon element
  9        ),
 10        SDO_ORDINATE_ARRAY (
 11           0,0, 2,0, 2,2, 0,2, 0,0     -- vertices of polygon
 12        )
 13     ),
 14     SDO_NUMBER_ARRAY(-1),              -- Just 1 ground height value applied to all
vertices
 15     SDO_NUMBER_ARRAY(1),               -- Just 1 top height value applied to all vertices
 16     'FALSE',                           -- No need to validate
 17     0.5                                -- Tolerance value
 18  ) EXTRUDED_GEOM FROM DUAL;

EXTRUDED_GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINA
--------------------------------------------------------------------------------
SDO_GEOMETRY(3008, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1007, 1, 1, 1006, 6, 1,
10
03, 1, 16, 1003, 1, 31, 1003, 1, 46, 1003, 1, 61, 1003, 1, 76, 1003, 1), SDO_ORD
INATE_ARRAY(0, 0, -1, 0, 2, -1, 2, 2, -1, 2, 0, -1, 0, 0, -1, 0, 0, 1, 2, 0, 1,
```

```
2, 2, 1, 0, 2, 1, 0, 0, 1, 0, 0, -1, 2, 0, -1, 2, 0, 1, 0, 0, 1, 0, 0, -1, 2, 0,
 -1, 2, 2, -1, 2, 2, 1, 2, 0, 1, 2, 0, -1, 2, 2, -1, 0, 2, -1, 0, 2, 1, 2, 2, 1,
 2, 2, -1, 0, 2, -1, 0, 0, -1, 0, 0, 1, 0, 2, 1, 0, 2, -1))
```

5.4 Validating Spatial Data

- **Validation Criteria**

  In a geometry, Oracle uses the SDO_ETYPE as a guide to determine whether a geometry is valid or invalid. Indeed, it is important to check whether the SDO_GEOMETRY data are in valid spatial format. Otherwise, you may get wrong results, errors, or failures when performing spatial queries.

  Point

  An example of using the VALIDATE_GEOMETRY_WITH_CONTEXT function to specifies diminfo. The advantage of using this function is to check whether all the coordinates are within the bounds specified in the diminfo attribute for basic validation.

  Using the diminfo Parameter in the VALIDATE_GEOMETRY_WITH_CONTEXT Function:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_GEOMETRY -- first argument to validate is geometry
  4     (
  5        2001, -- point type
  6        NULL,
  7        SDO_POINT_TYPE(-80,20,NULL), -- point is <-80,20> and is out of range.
  8        NULL,
  9        NULL
 10     ),
 11     SDO_DIM_ARRAY -- second argument is diminfo (of type SDO_DIM_ARRAY)
 12     (
 13        SDO_DIM_ELEMENT('X', 0, 50, 0.5), -- lower, upper bound range is 0 to 50
 14        SDO_DIM_ELEMENT('Y', 0, 50, 0.5) -- lower, upper bound range is 0 to 50
 15     )
 16  ) is_valid FROM DUAL;

IS_VALID
--------------------------------------------------------------------------------
13011 -- Coordinate value out of dimension range
```

  In this example:

  - considering the point geometry with longitude=–80 and latitude=20
  - if the diminfo is set to (0, 50) for both dimensions, the point will be invalid

- the ORA-13011 error: the longitude value of −80 is out of range (0 to 50) for that dimension

Below is an example of using the tolerance Parameter in the VALIDATE_GEOMETRY_WITH_CONTEXT Function.

Using the tolerance Parameter in the:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_GEOMETRY -- first argument to validate is geometry
  4     (
  5        2001, -- point type
  6        NULL,
  7        SDO_POINT_TYPE(-80,20,NULL), --point not out of range as no range specified
  8        NULL,
  9        NULL
 10     ),
 11     0.5
 12  ) is_valid FROM DUAL;

IS_VALID
--------------------------------------------------------------------------------
TRUE
```

String

When validating a line string, a rules need to be follow:

- all points in the line are distinct
- a line should have two or more points

Below is an example of the result of validating a line string with duplicate points.

Validating a Line String with Duplicate Points:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_GEOMETRY -- first argument to validate is geometry
  4     (
  5        2002, -- Line String type
  6        NULL,
  7        NULL,
  8        SDO_ELEM_INFO_ARRAY(1,2,1), -- Line String
  9        SDO_ORDINATE_ARRAY (
 10           1,1, -- first vertex
 11           2,2, -- second vertex
 12           2,2 -- third vertex: same as second
 13        )
 14     ),
 15     0.5 -- second argument: tolerance
```

```
 16   ) is_valid FROM DUAL;

IS_VALID
--------------------------------------------------------------------------------
13356 [Element <1>] [Coordinate <2>]
```

Polygon

Polygons define a contiguous area bounded by one outer ring on the exterior and by zero or more inner rings on the interior.

Below is an example of validating a invalid polygon.

an invalid polygon: edges of the ring of the polygon cross each other

Validation on a Self-Crossing Geometry:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_GEOMETRY
  4     (
  5        2003,              -- 2-D Polygon
  6        NULL,
  7        NULL,
  8        SDO_ELEM_INFO_ARRAY
  9        (
 10           1, 1003,1     -- Polygonal ring connected by lines
 11        ),
 12        SDO_ORDINATE_ARRAY
 13        (
 14           2,2,           -- first vertex
 15           3,3.5,         -- second vertex. Edge 1 is between previous and this vertex.
 16           2,5,
 17           5,5,
 18           3,3.5,         -- fifth vertex. Edge 4 is between previous and this vertex.
 19           5,2,
 20           2,2
 21        )
 22     ),
 23     0.000005
 24  )
 25  FROM dual;
```

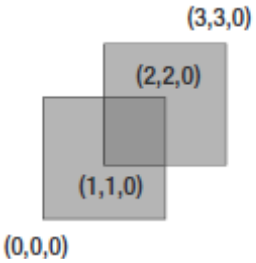
```
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(SDO_GEOMETRY(2003,--2-DPOL
YGONNULL,NULL,
--------------------------------------------------------------------------------
13349 [Element <1>] [Ring <1>][Edge <1>][Edge <4>]
```

In this example:

- the return values shows:
  - element 1 is invalid
  - edge 1 connecting (2, 2) and (3, 3.5)
  - edge 4 connecting (5, 5) and (3, 3.5)
  - the polygon boundary crosses itself

Composite Surface

Below is an example of validating the Composite Surface.



Examples of invalid composite surfaces: non-overlapping polygon rule

Validation on the Composite Surface:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_GEOMETRY -- first argument to validate is geometry
  4     (
  5        3003, -- 3-D Polygon/Surface type
  6        NULL,
  7        NULL,
  8        SDO_ELEM_INFO_ARRAY(
  9           1, 1006, 2, -- Composite Surface with 2 Polygons
 10           1, 1003, 1, -- 1st polygon
 11           16, 1003, 1 -- 2nd polygon
 12        ),
 13        SDO_ORDINATE_ARRAY (
 14           0,0,0, 2,0,0, 2,2,0, 0,2,0, 0,0,0, -- vertices of first polygon
 15           1,1,0, 3,1,0, 3,3,0, 1,3,0, 1,1,0 -- vertices of second polygon
 16        )
 17     ),
 18  0.5 -- second argument: tolerance
 19  ) is_valid FROM DUAL;

IS_VALID
```

```
--------------------------------------------------------------------------------
54515 Point:0,Edge:2,Ring:1,Polygon:1,
```

In this example:

- the return values shows:
  - ORA-54515: Outer rings in a composite surface intersect
  - the second edge from vertex (2,0,0) to vertex (2,2,0) from first polygon intersecting with another edge

Simple Solid

Below is an example of validating the Simple Solid.

Validating the Simple Solid:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_GEOMETRY(3008, NULL, NULL,
  4        SDO_ELEM_INFO_ARRAY(
  5           1, 1007, 1, -- Solid element
  6           1, 1006, 5, -- Composite surface with 5 polygons
  7           1, 1003,1, 16, 1003, 1, 31, 1003, 1, 46, 1003, 1, 61, 1003, 1
  8        ),
  9        SDO_ORDINATE_ARRAY(
 10           0, 0, 0, 0, 4, 0, 4, 4, 0, 4, 0, 0, 0, 0, 0,
 11           4, 4, 4, 0, 4, 4, 0, 0, 4, 4, 0, 4, 4, 4, 4,
 12           0, 0, 0, 4, 0, 0, 4, 0, 4, 0, 0, 4, 0, 0, 0,
 13           0, 0, 0, 0, 0, 4, 0, 4, 4, 0, 4, 0, 0, 0, 0,
 14           4, 4, 4, 4, 0, 4, 4, 0, 0, 4, 4, 0, 4, 4, 4
 15        )
 16     ),
 17  0.5
 18  ) is_valid FROM DUAL;

IS_VALID
--------------------------------------------------------------------------------
54502 Point:0,Edge:2,Ring:1,Polygon:1,Comp-Surf:1,
```

- **Collections**

For collections of multiple elements, the VALIDATE_LAYER_WITH_CONTEXT function can be used to perform validation on a single geometry.

Using the VALIDATE_LAYER_WITH_CONTEXT Procedure:

```
-- create validate_results table to store the results
SQL> CREATE TABLE validate_results(sdo_rowid ROWID, status VARCHAR2(2000));

Table created.

-- create VALIDATE_LAYER_WITH_CONTEXT Procedure
```

```
SQL> BEGIN
  2   SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
  3   (
  4      'SALES_REGIONS',
  5      'GEOM',
  6      'VALIDATE_RESULTS'
  7   );
  8   END;
  9   /

PL/SQL procedure successfully completed.

-- provide checking for the procedure
SQL> SELECT * FROM validate_results;

SDO_ROWID STATUS
------------------- --------------------------------------------------
AAALctAADAAATRvAAA 13356 [Element <1>] [Coordinate <17>][Ring <1>]
```

5.5 Debugging Spatial Data

Oracle Spatial provides a number of functions in SDO_UTIL package to debug and clean data loaded into an SDO_GEOMETRY column, such as REMOVE_DUPLICATE_VERTICES function, VALIDATE_GEOMETRY_WITH_CONTEXT function,

- **REMOVE_DUPLICATE_VERTICES**
  Below is an example of using the REMOVE_DUPLICATE_VERTICES function to remove duplicate vertices from an SDO_GEOMETRY object.
  Example of Removing Duplicate Vertices in a Geometry:

```
SQL>  SELECT  geom,  SDO_UTIL.REMOVE_DUPLICATE_VERTICES(sr.geom,0.5)
nodup_geom
  2   FROM sales_regions sr
  3   WHERE id=1000;

GEOM
------------
SDO_GEOMETRY
(
2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARRAY
(
```

```
-77.04487, 38.9043742, -77.046645, 38.9040983, -77.04815, 38.9033127,
-77.049155, 38.9021368, -77.049508, 38.9007499, -77.049155, 38.899363,
-77.048149, 38.8981873, -77.046645, 38.8974017, -77.04487, 38.8971258,
-77.043095, 38.8974017, -77.041591, 38.8981873, -77.040585, 38.899363,
-77.040232, 38.9007499, -77.040585, 38.9021368, -77.04159, 38.9033127,
-77.043095, 38.9040983, -77.04487, 38.9043742, -77.04487, 38.9043742
)
)
NODUP_GEOM
----------------------
SDO_GEOMETRY
(
2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARRAY
(
-77.04487, 38.9043742, -77.046645, 38.9040983, -77.04815, 38.9033127,
-77.049155,38.9021368, -77.049508, 38.9007499, -77.049155, 38.899363,
-77.048149, 38.8981873,-77.046645, 38.8974017, -77.04487, 38.8971258,
-77.043095, 38.8974017, -77.041591,38.8981873, -77.040585, 38.899363,
-77.040232, 38.9007499, -77.040585, 38.9021368, -77.04159, 38.9033127,
-77.043095, 38.9040983, -77.04487, 38.9043742
)
)
```

In this example:

- the first point (at −77.04487, 38.9043742) and the last point (at −77.04487, 38.9043742) are the same

Below is an example of rerun the VALIDATE_GEOMETRY_WITH_CONTEXT function on this result geometry.

Validating After Removing the Duplicate Vertices:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_UTIL.REMOVE_DUPLICATE_VERTICES(a.geom, 0.5),
  4     0.5
  5  ) is_valid
  6  FROM sales_regions a
  7  WHERE id=10000;

IS_VALID
--------------------------------------------------------------------------------
TRUE
```

- **EXTRACT**

Below is an example of an example extracting the second element of a multi-polygon

geometry.

Extracting the Second Element from a Geometry:

```
SQL> SELECT SDO_UTIL.EXTRACT
  2  (
  3    SDO_GEOMETRY
  4    (
  5      2007, -- multipolygon collection type geometry
  6      NULL,
  7      NULL,
  8      SDO_ELEM_INFO_ARRAY
  9      (
 10        1,1003,3, -- first element descriptor triplet: for rectangle polygon
 11        -- (see Figure 4-10 and the accompanying listing in Chapter 4)
 12        5, 1003, 1 -- second element descriptor triplet:
 13        -- starting offset 5 means it starts at the 5th ordinate
 14      ),
 15      SDO_ORDINATE_ARRAY
 16      (
 17        1,1,2,2, -- first element ordinates (four for mbr)
 18        3,3, 4, 3, 4,4, 3,4, 3,4,3,3 -- second element starting at 5th ordinate:
 19        -- this second element is returned
 20      )
 21    ), -- End of the Geometry
 22    2 -- specifies the element number to extract
 23  ) second_elem
 24  FROM dual;

SECOND_ELEM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATE
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARRAY(3, 3, 4, 3, 4, 4, 3, 4, 3, 4, 3, 3))
```

Below is an example of performing validation on the specific element to identify what is wrong with it, which is used to check on the result of previous example.

Validation of an Extracted Geometry:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3    SDO_UTIL.EXTRACT
  4    (
  5      SDO_GEOMETRY
  6      (
  7        2007, null, null,
  8        SDO_ELEM_INFO_ARRAY(1,1003,3, 5, 1003, 1),
```

```
 9           SDO_ORDINATE_ARRAY
10           (
11              1,1,2,2, -- first element of multipolygon geometry
12              3,3, 4, 3, 4,4, 3,4, 3,4,3,3 -- second element of multipolygon geometry
13           )
14        ),
15        2 -- element number to extract
16     ),
17     0.00005
18  )
19  FROM dual;

SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(SDO_UTIL.EXTRACT(SDO_GEO
METRY(2007,NULL,
--------------------------------------------------------------------------------
13356 [Element <1>] [Coordinate <4>][Ring <1>]
```

Below is an example of validating the result of SDO_UTIL.EXTRACT in different format.

Validation on the Result of SDO_UTIL.EXTRACT:

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  2  (
  3     SDO_GEOMETRY
  4     (
  5        2003, NULL, NULL,
  6        SDO_ELEM_INFO_ARRAY(1, 1003, 1),
  7        SDO_ORDINATE_ARRAY
  8        (
  9           3, 3, -- first vertex coordinates
 10           4, 3, -- second vertex coordinates
 11           4, 4, -- third vertex coordinates
 12           3, 4, -- fourth vertex coordinates
 13           3, 4, -- fifth vertex coordinates
 14           3, 3   -- sixth vertex coordinates (same as first for polygon)
 15        )
 16     ),
 17     0.00005 -- tolerance
 18  ) FROM dual;

SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(SDO_GEOMETRY(2003,NULL,NU
LL,SDO_ELEM_INF
--------------------------------------------------------------------------------
13356 [Element <1>] [Coordinate <4>][Ring <1>]
```

In the above two examples:

- The result of validation are the same: 13356 [Element <1>] [Coordinate <4>][Ring <1>]
- 13356 <Coordinate 4>: indicates a duplicate vertex at the fourth (and fifth) vertex coordinates of the SDO_ORDINATE_ARRAY5
  - the ordinate array is (3, 3, 4, 3, 4, 4, 3, 4, 3, 4, 3, 3)
  - the fourth(3, 4) and fifth(3, 4) vertexes are duplicates

Below is an example of using REMOVE_DUPLICATE_VERTICES function to remove the duplicate coordinate.

Removing Duplicate Vertices:

```
SQL> SELECT SDO_UTIL.REMOVE_DUPLICATE_VERTICES
  2  (
  3     SDO_UTIL.EXTRACT
  4     (
  5        SDO_GEOMETRY
  6        (
  7           2007, NULL, NULL,
  8           SDO_ELEM_INFO_ARRAY(1,1003,3, 5, 1003, 1),
  9           SDO_ORDINATE_ARRAY
 10           (
 11              1,1,2,2,
 12              3,3, 4, 3, 4,4, 3,4, 3,4,3,3
 13           )
 14        ),
 15        2
 16     ),
 17     0.00005
 18  )
 19  FROM dual;

SDO_UTIL.REMOVE_DUPLICATE_VERTICES(SDO_UTIL.EXTRACT(SDO_GEOMETRY(20
07,NULL,NULL,
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARRAY(3, 3, 4, 3, 4, 4, 3, 4, 3, 3))
```

In this example:

- the fourth vertex coordinates which is duplicate (3,4) at fifth that is removed

- **APPEND**

The SDO_UTIL.APPEND function can combine multiple geometries as long as they do not intersect, and takes two geometries and a tolerance and appends them into a single geometry. Below is an example of using the SDO_UTIL.APPEND function.

Example of SDO_UTIL.APPEND:

```
SQL> SELECT
  2  SDO_UTIL.APPEND
  3  (
  4    SDO_UTIL.EXTRACT
  5    (
  6      SDO_GEOMETRY
  7      (
  8        2007, NULL, NULL,
  9        SDO_ELEM_INFO_ARRAY(
 10          1,1003,3, -- First element is as a rectangle polygon
 11          5, 1003, 1
 12        ),
 13        SDO_ORDINATE_ARRAY(
 14          1,1, 2,2,
 15          3,3, 4,3,
 16          4,4, 3,4,
 17          3,4, 3,3
 18        )
 19      ),
 20      1
 21    ),
 22    SDO_UTIL.REMOVE_DUPLICATE_VERTICES
 23    (
 24      SDO_GEOMETRY
 25      (
 26        2007, NULL, NULL,
 27        SDO_ELEM_INFO_ARRAY(1,1003,3, 5, 1003, 1),
 28        SDO_ORDINATE_ARRAY
 29        (
 30          1,1, 2,2,
 31          3,3, 4,3,
 32          4,4, 3,4,
 33          3,4, 3,3
 34        )
 35      ),
 36      0.00005
 37    )
 38  ) combined_geom
 39  FROM dual;

COMBINED_GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINA
--------------------------------------------------------------------------------
SDO_GEOMETRY(2007, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3, 5, 1003, 1, 15,
```

```
1003, 1), SDO_ORDINATE_ARRAY(1, 1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 1, 1, 3, 3, 4, 3, 4, 4, 3, 4, 3,
3))
```

In this example:

- the function firstly specifies the element as a rectangle polygon using the triplet <1,1003,3> in SDO_ELEM_INFO_ARRAY to five vertices for the polygon
- then performs an append of the vertices of the first element and the second element, removing any duplicates automatically in the process

- **GETNUMELEM and GETNUMVERTICES**

Below are two examples of using GETNUMELEM and GETNUMVERTICES functions to inspect the number of elements or vertices or get the set of vertices in an SDO_GEOMETRY object.

Finding the Number of Elements in a Geometry:

```
-- use GETNUMELEM function
SQL> SELECT SDO_UTIL.GETNUMELEM(geom) nelem
  2  FROM sales_regions
  3  WHERE id=10000;

     NELEM
----------
         1

-- use GETNUMVERTICES function
SQL> SELECT SDO_UTIL.GETNUMVERTICES(geom) nverts
  2  FROM sales_regions
  3  WHERE id=10000;

    NVERTS
----------
        17
```

- **EXTRACT3D**

Below is an example of using EXTRACT3D function to identify the edge that caused the problem in a Simple Solid (discussed in page 63), and the result shows the edge that is not closed is the edge from (0, 4, 0) to (4, 4, 0).

Extracting the Invalid Edge:

```
SQL> SELECT SDO_UTIL.EXTRACT3D
  2  (
  3     SDO_GEOMETRY(3008, NULL, NULL,
  4     SDO_ELEM_INFO_ARRAY(
  5        1, 1007, 1, -- Solid element
  6        1, 1006, 5, -- Composite surface with 5 polygons
  7        1, 1003,1, 16, 1003, 1, 31, 1003, 1, 46, 1003, 1, 61, 1003, 1
```

```
 8      ),
 9      SDO_ORDINATE_ARRAY(
10         0, 0, 0, 0, 4, 0, 4, 4, 0, 4, 0, 0, 0, 0, 0, -- Vertices of 2nd edge bold
11         4, 4, 4, 0, 4, 4, 0, 0, 4, 4, 0, 4, 4, 4, 4,
12         0, 0, 0, 4, 0, 0, 4, 0, 4, 0, 0, 4, 0, 0, 0,
13         0, 0, 0, 0, 0, 4, 0, 4, 4, 0, 4, 0, 0, 0, 0,
14         4, 4, 4, 4, 0, 4, 4, 0, 0, 4, 4, 0, 4, 4, 4
15      )
16   ),
17   '0,2,1,1,1' -- LABEL String for extracting the
18   -----2nd edge of Ring1, Polygon1,Comp Surface1
19   ) edge FROM DUAL;

EDGE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(3002, NULL, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1),
SDO_ORDINATE_ARRAY(0, 4, 0, 4, 4, 0))
```

6.0 GEOCODING

In this section, the concepts of geocoding and geocoding process are used to examine the functionality of the geocoder in Oracle Spatial. Firstly, the method of converting street addresses into geographical locations is examined to location-enable data for spatial searches and various analyses. In addition, this is the first step in adding spatial intelligence to an application, and the test of using geocoder has done to correct and clean errors in the input addresses. On the other hand, the spatial catalog is used to determine and extrapolate the location for a specified address.

6.1 Geocoding

In general, there are two purposes of using Geocoding serves, which are:

- to associate geographical coordinates with an address
- to correct various errors in addresses

Below is an example of an example of getting the coordinates from an address using the simple GEOCODE_AS_GEOMETRY function that returns a point SDO_GEOMETRY object. That object contains the geographical coordinates that the geocoder associated with this address.

Geocoding an Address:

```
SQL> SELECT SDO_GCDR.GEOCODE_AS_GEOMETRY
  2  (
  3    'TEST',
  4    SDO_KEYWORDARRAY
  5    (
  6      '3746 CONNECTICUT AVE NW',
  7      'WASHINGTON, DC 20008'
  8    ),
  9    'US'
 10  ) geom
 11  FROM DUAL ;

GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-77.060283, 38.9387083, NULL), NULL,
NULL)
```

Below is an example of obtaining corrections for a misspelled address using the GEOCODE function.

Geocoding and Normalizing an Address:

```
SQL> SELECT SDO_GCDR.GEOCODE
  2  (
  3     'TEST',
  4     SDO_KEYWORDARRAY
  5     (
  6        '3746 CONNECTICT AVE NW',
  7        'WASHINGTON, DC 20348'
  8     ),
  9     'US',
 10     'DEFAULT'
 11  ) geom
 12  FROM DUAL ;

GEOM(ID, ADDRESSLINES, PLACENAME, STREETNAME, INTERSECTSTREET, SECUNIT,
SETTLEME
--------------------------------------------------------------------------------
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), NULL, 'CONNECTICUT AVE NW', NULL,
NULL, 'WASHINGTON', NULL, 'DC', 'US', '20008', NULL, '20008', NULL, '3746',
'CONNECTICUT', 'AVE', 'F', 'F', NULL, 'NW', 'L', .944444444, 18571166, '????#E?UT?B281C??',
10, 'DEFAULT', -77.060283, 38.9387083, '????0120010??002?')
```

6.2 Using Geocoder Functions

- **GEOCODE_AS_GEOMETRY**

  This function returns an SDO_GEOMETRY object with the corresponding geographical location for that address.

  Below is an example of geocoding a street address in San Francisco using GEOCODE_AS_GEOMETRY function.

  Using the GEOCODE_AS_GEOMETRY Function:

```
SQL> SELECT SDO_GCDR.GEOCODE_AS_GEOMETRY
  2  (
  3     'TEST',
  4     SDO_KEYWORDARRAY('1250 Clay Street', 'San Francisco, CA'),
  5     'US'
  6  )
  7  FROM DUAL;

SDO_GCDR.GEOCODE_AS_GEOMETRY('TEST',SDO_KEYWORDARRAY('1250CLAYSTR
EET','SANFRANCI
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.41356, 37.7932878, NULL), NULL,
NULL)
```

Below is an example of geocoding a location where the house number does not exist (the highest house number on Clay Street is 3999).

Using the GEOCODE_AS_GEOMETRY Function with an Invalid House Number:

```
SQL> SELECT SDO_GCDR.GEOCODE_AS_GEOMETRY
  2  (
  3    'TEST',
  4    SDO_KEYWORDARRAY('4500 Clay Street', 'San Francisco, CA'),
  5    'US'
  6  )
  7  FROM DUAL;

SDO_GCDR.GEOCODE_AS_GEOMETRY('TEST',SDO_KEYWORDARRAY('4500CLAYSTR
EET','SANFRANCI
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.41437, 37.79318, NULL), NULL,
NULL)
```

In this example:

- the result is still a valid geometry pointing to a house on the street
- there are no indication of the exact house on which the geocoder positioned the coordinates

Below is an example of geocoding a location where the street number does not exist

Using the GEOCODE_AS_GEOMETRY Function with an Invalid Street Name:

```
SQL> SELECT SDO_GCDR.GEOCODE_AS_GEOMETRY
  2  (
  3    'TEST',
  4    SDO_KEYWORDARRAY('Cloy Street', 'San Francisco, CA'),
  5    'US'
  6  )
  7  FROM DUAL;

SDO_GCDR.GEOCODE_AS_GEOMETRY('TEST',SDO_KEYWORDARRAY('CLOYSTREET',
'SANFRANCISCO,
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(0, 0, NULL), NULL, NULL)
```

In this example:

- the result return a location in the middle of the town (or postal code if one was specified)

- **GEOCODE**

A Street Address Without a House Number

Below is an example of geocoding a street address in San Francisco with a specified street name and town but no postal code.

Example of Calling the GEOCODE Function:

```
SQL> SELECT SDO_GCDR.GEOCODE
  2  (
  3    'TEST',
  4    SDO_KEYWORDARRAY('Clay Street', 'San Francisco, CA'),
  5    'US',
  6    'DEFAULT'
  7  )
  8  FROM DUAL;

SDO_GCDR.GEOCODE('TEST',SDO_KEYWORDARRAY('CLAYSTREET','SANFRANCISCO
,CA'),'US','D
--------------------------------------------------------------------------------
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(NULL), NULL, 'CLAY ST', NULL, NULL, 'SAN
FRANCISCO', NULL, 'CA', 'US', '94108', NULL, '94108', NULL, '978', 'CLAY', 'ST', 'F',
'F', NULL, NULL, 'L', 0, 1, 23600689, 'nul?#ENUT?B281CP?', 1, 'DEFAULT', -122.40904,
37.79385)
```

An example of creating Procedure FORMAT_GEO_ADDR takes an SDO_GEO_ADDR object as input and formats it using the DBMS_OUTPUT package to format the display.

FORMAT_GEO_ADDR Procedure:

```
SQL> CREATE OR REPLACE PROCEDURE format_geo_addr (
  2    address SDO_GEO_ADDR
  3  )
  4  AS
  5  BEGIN
  6    dbms_output.put_line ('- ID                    ' || address.ID);
  7    dbms_output.put_line ('- ADDRESSLINES');
  8    if address.addresslines.count() > 0 then
  9      for i in 1..address.addresslines.count() loop
 10        dbms_output.put_line  ('-  ADDRESSLINES['||i||']          ' ||
address.ADDRESSLINES(i));
 11      end loop;
 12    end if;
 13    dbms_output.put_line ('- PLACENAME             ' || address.PLACENAME);
 14    dbms_output.put_line ('- STREETNAME            ' || address.STREETNAME);
 15      dbms_output.put_line  ('-  INTERSECTSTREET          ' ||
address.INTERSECTSTREET);
 16    dbms_output.put_line ('- SECUNIT               ' || address.SECUNIT);
```

```
 17      dbms_output.put_line ('- SETTLEMENT' || address.SETTLEMENT);
 18      dbms_output.put_line ('- MUNICIPALITY' || address.MUNICIPALITY);
 19      dbms_output.put_line ('- REGION' || address.REGION);
 20      dbms_output.put_line ('- COUNTRY' || address.COUNTRY);
 21      dbms_output.put_line ('- POSTALCODE ' || address.POSTALCODE);
 22      dbms_output.put_line ('-POSTALADDONCODE' || address.POSTALADDONCODE);
 23      dbms_output.put_line ('-FULLPOSTALCODE' || address.FULLPOSTALCODE);
 24      dbms_output.put_line ('- POBOX' || address.POBOX);
 25      dbms_output.put_line ('- HOUSENUMBER' || address.HOUSENUMBER);
 26      dbms_output.put_line ('- BASENAME' || address.BASENAME);
 27      dbms_output.put_line ('- STREETTYPE' || address.STREETTYPE);
 28      dbms_output.put_line ('- STREETTYPEBEFORE' || address.STREETTYPEBEFORE);
 29          dbms_output.put_line ('- STREETTYPEATTACHED' ||
    address.STREETTYPEATTACHED);
 30      dbms_output.put_line ('- STREETPREFIX'|| address.STREETPREFIX);
 31      dbms_output.put_line ('- STREETSUFFIX' || address.STREETSUFFIX);
 32      dbms_output.put_line ('- SIDE' || address.SIDE);
 33      dbms_output.put_line ('- PERCENT' || address.PERCENT);
 34      dbms_output.put_line ('- EDGEID' || address.EDGEID);
 35      dbms_output.put_line ('-ERRORMESSAGE' || address.ERRORMESSAGE);
 36      dbms_output.put_line ('- MATCHVECTOR' || address.MATCHVECTOR);
 37       dbms_output.put_line ('-    '|| substr (address.errormessage,5,1)   ||'  '||substr
(address.matchvector,5,1)   ||' House or building number');
 38       dbms_output.put_line ('-    '|| substr (address.errormessage,6,1)   ||'  '||substr
(address.matchvector,6,1)   ||' Street prefix');
 39       dbms_output.put_line ('-    '|| substr (address.errormessage,7,1)   ||'  '||substr
(address.matchvector,7,1)   ||' Street base name');
 40       dbms_output.put_line ('-    '|| substr (address.errormessage,8,1)   ||'  '||substr
(address.matchvector,8,1)   ||' Street suffix');
 41       dbms_output.put_line ('-    '|| substr (address.errormessage,9,1)   ||'  '||substr
(address.matchvector,9,1)   ||' Street type');
 42       dbms_output.put_line ('-    '|| substr (address.errormessage,10,1)  ||'  '||substr
(address.matchvector,10,1) ||' Secondary unit');
 43       dbms_output.put_line ('-    '|| substr (address.errormessage,11,1)  ||'  '||substr
(address.matchvector,11,1) ||' Built-up area or city');
 44       dbms_output.put_line ('-    '|| substr (address.errormessage,14,1)  ||'  '||substr
(address.mat
chvector,14,1) ||' Region');
 45       dbms_output.put_line ('-    '|| substr (address.errormessage,15,1)  ||'  '||substr
(address.mat
chvector,15,1) ||' Country');
 46       dbms_output.put_line ('-    '|| substr (address.errormessage,16,1)  ||'  '||substr
(address.mat
chvector,16,1) ||' Postal code');
```

```
 47       dbms_output.put_line ('-       '||    substr   (address.errormessage,17,1)||'  '||substr
(address.matchvector,17,1) ||' Postal add-on code');
 48     dbms_output.put_line ('- MATCHCODE                    ' || address.MATCHCODE || ' =
' ||
 49        case address.MATCHCODE
 50           when   1 then 'Exact match'
 51           when   2 then 'Street type not matched'
 52           when   3 then 'House number not matched'
 53           when   4 then 'Street name not matched'
 54           when 10 then 'Postal code not matched'
 55           when 11 then 'City not matched'
 56        end
 57     );
 58     dbms_output.put_line ('- MATCHMODE                     ' || address.MATCHMODE);
 59     dbms_output.put_line ('- LONGITUDE                    ' || address.LONGITUDE);
 60     dbms_output.put_line ('- LATITUDE                    ' || address.LATITUDE);
 61   end;
 62   /

Procedure created.
```

Below is an example of using the FORMAT_GEO_ADDR Procedure with the previous example

Example of Using the FORMAT_GEO_ADDR Procedure:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2     FORMAT_GEO_ADDR (
  3        SDO_GCDR.GEOCODE (
  4           'TEST',
  5           SDO_KEYWORDARRAY('Clay Street', 'San Francisco, CA
  6           'US',
  7           'DEFAULT'
  8        )
  9     );
 10   END;
 11   /
- ID                            0
- ADDRESSLINES
- PLACENAME
- STREETNAME                    CLAY ST
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                    SAN FRANCISCO
- MUNICIPALITY
```

```
- REGION                    CA
- COUNTRY                    US
- POSTALCODE                 94108
- POSTALADDONCODE
- FULLPOSTALCODE             94108
- POBOX
- HOUSENUMBER                978
- BASENAME                  CLAY
- STREETTYPE                ST
- STREETTYPEBEFORE           F
- STREETTYPEATTACHED         F
- STREETPREFIX
- STREETSUFFIX
- SIDE                      L
- PERCENT                    0
- EDGEID                    23600689
- ERRORMESSAGE               ????#ENUT?B281CP?
- MATCHVECTOR                ????4101010??004?
-    # 4 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    T 0 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                  1 = Exact match
- MATCHMODE                  DEFAULT
- LONGITUDE                 -122.40904
- LATITUDE                  37.79385

PL/SQL procedure successfully completed.
```

In this example:

- a geographical point received on Clay Street
- the received address is in:
  - street name: CLAY ST
  - ZIP code: 94108
  - house number: 978
- the MATCHCODE returned is 1: indicating we had a full match

Dissecting Clay Street

Below example shows how to find out the house numbers for a street.

Getting Street Details from the Geocode Reference Data:

```
SQL> SELECT road_id, name, postal_code, start_hn, center_hn, end_hn
  2  FROM     gc_road_us
  3  WHERE    name = 'CLAY ST'
  4  AND      postal_code like '94%'
  5  ORDER BY start_hn;

ROAD_ID NAME     POSTAL     START_HN     CENTER_HN     END_HN
---------- ---------------------------------------------------------- ------- ---------- ---------- ----------
 767 CLAY ST         94111            1              398              699
 427 CLAY ST         94108          700              978             1299
 505 CLAY ST         94109         1300             1698             1999
1213 CLAY ST         94115         2200             2798             3299
1446 CLAY ST         94118         3300             3698             3999
```

In this example:

- the geocoder picked the one with the smallest number (94108) and then the center house number (978)

A Street Address with a House Number

Below example shows how to find out the house numbers for a street without a specified the street type.

Using the GEOCODE Function with aValid House Number:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2     FORMAT_GEO_ADDR (
  3        SDO_GCDR.GEOCODE (
  4        'TEST',
  5        SDO_KEYWORDARRAY('1350 Clay', 'San Francisco, CA'),
  6        'US',
  7        'DEFAULT'
  8        )
  9     );
 10  END;
 11  /
- ID                            0
- ADDRESSLINES
- PLACENAME
- STREETNAME                    CLAY ST
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                    SAN FRANCISCO
- MUNICIPALITY
```

```
- REGION                    CA
- COUNTRY                    US
- POSTALCODE                 94109
- POSTALADDONCODE
- FULLPOSTALCODE             94109
- POBOX
- HOUSENUMBER                1350
- BASENAME                  CLAY
- STREETTYPE                ST
- STREETTYPEBEFORE           F
- STREETTYPEATTACHED         F
- STREETPREFIX
- STREETSUFFIX
- SIDE                      L
- PERCENT                    .49
- EDGEID                    23600696
- ERRORMESSAGE               ????#ENU??B281CP?
- MATCHVECTOR                ????0101410??004?
-    # 0 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    ? 4 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                  2 = Street type not matched
- MATCHMODE                  DEFAULT
- LONGITUDE                 -122.4152166
- LATITUDE                  37.7930729

PL/SQL procedure successfully completed.
```

In this example:

- the MATCHCODE returned is 2, which means there is no match on the street type

Correcting Invalid Address

Below example shows how to find a street with an invalid house number.

Using the GEOCODE Function with an Invalid House Number:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
```

```
  2      FORMAT_GEO_ADDR (
  3        SDO_GCDR.GEOCODE (
  4          'TEST',
  5          SDO_KEYWORDARRAY('4500 Clay Street', 'San Francisco, CA'),
  6          'US',
  7          'DEFAULT'
  8        )
  9     );
 10   END;
 11   /
- ID                            0
- ADDRESSLINES
- PLACENAME
- STREETNAME                    CLAY ST
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                    SAN FRANCISCO
- MUNICIPALITY
- REGION                        CA
- COUNTRY                       US
- POSTALCODE                    94108
- POSTALADDONCODE
- FULLPOSTALCODE                94108
- POBOX
- HOUSENUMBER                   1299
- BASENAME                      CLAY
- STREETTYPE                    ST
- STREETTYPEBEFORE              F
- STREETTYPEATTACHED            F
- STREETPREFIX
- STREETSUFFIX
- SIDE                          R
- PERCENT                       0
- EDGEID                        23600695
- ERRORMESSAGE                  ?????ENUT?B281CP?
- MATCHVECTOR                   ????2101010??004?
-    ? 2 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    T 0 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
```

```
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                    3 = House number not matched
- MATCHMODE                    DEFAULT
- LONGITUDE                   -122.41437
- LATITUDE                    37.79318

PL/SQL procedure successfully completed.
```

In this example:

- the MATCHCODE returned is 3, which means there is no match on the house number
- the coordinates returned the highest house number in the first segment of the street: house 1299 in postal code 94108

Below example shows how to find a street with an invalid postcode.

Using the GEOCODE Function with an Invalid Postal Codes:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2     FORMAT_GEO_ADDR (
  3        SDO_GCDR.GEOCODE (
  4           'TEST',
  5           SDO_KEYWORDARRAY('1350 Clay St', 'San Francisco, CA 99130'),
  6           'US',
  7           'DEFAULT'
  8        )
  9     );
 10  END;
 11  /
- ID                          0
- ADDRESSLINES
- PLACENAME
- STREETNAME                  CLAY ST
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                  SAN FRANCISCO
- MUNICIPALITY
- REGION                      CA
- COUNTRY                     US
- POSTALCODE                  94109
- POSTALADDONCODE
- FULLPOSTALCODE              94109
- POBOX
- HOUSENUMBER                 1350
```

```
- BASENAME                      CLAY
- STREETTYPE                    ST
- STREETTYPEBEFORE              F
- STREETTYPEATTACHED            F
- STREETPREFIX
- STREETSUFFIX
- SIDE                          L
- PERCENT                       .49
- EDGEID                        23600696
- ERRORMESSAGE                  ????#ENUT?B281C??
- MATCHVECTOR                   ????0101010??002?
-    # 0 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    T 0 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    ? 2 Postal code
-    ? ? Postal add-on code
- MATCHCODE                      10 = Postal code not matched
- MATCHMODE                      DEFAULT
- LONGITUDE                     -122.4152166
- LATITUDE                      37.7930729


PL/SQL procedure successfully completed.
```

In this example:

- the MATCHCODE returned is 10, which means there is no match on the post code
- the coordinates are correctly positioned on number 1350 Clay Street, and the correct postal code (94109)

Using the EXACT Match Mode

Below example shows how to find a street with an invalid post code using EXACT mode.

Using the GEOCODE Function with an Invalid Postal Code (in EXACTMode):

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2     FORMAT_GEO_ADDR (
  3          SDO_GCDR.GEOCODE (
  4          'TEST',
  5          SDO_KEYWORDARRAY('1350 Clay St', 'San Francisco, CA 991
```

```
  6           'US',
  7           'EXACT'
  8         )
  9      );
 10   END;
 11   /
- ID                            0
- ADDRESSLINES
- PLACENAME
- STREETNAME
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                    SAN FRANCISCO
- MUNICIPALITY
- REGION                        CA
- COUNTRY                       US
- POSTALCODE
- POSTALADDONCODE
- FULLPOSTALCODE
- POBOX
- HOUSENUMBER
- BASENAME
- STREETTYPE
- STREETTYPEBEFORE              F
- STREETTYPEATTACHED            F
- STREETPREFIX
- STREETSUFFIX
- SIDE                          L
- PERCENT                       0
- EDGEID                        0
- ERRORMESSAGE                  ??????????B281C??
- MATCHVECTOR                   ????3131310??003?
-     ? 3 House or building number
-     ? 1 Street prefix
-     ? 3 Street base name
-     ? 1 Street suffix
-     ? 3 Street type
-     ? 1 Secondary unit
-     B 0 Built-up area or city
-     1 0 Region
-     C 0 Country
-     ? 3 Postal code
-     ? ? Postal add-on code
- MATCHCODE                      10 = Postal code not matched
```

```
- MATCHMODE                    EXACT
- LONGITUDE                   0
- LATITUDE                    0

PL/SQL procedure successfully completed.
```

Geocoding on Business Name

Below is an example of finding the location and address of the Transamerica Pyramid in San Francisco.

Using the GEOCODE Function to Find a POI:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2     FORMAT_GEO_ADDR (
  3        SDO_GCDR.GEOCODE (
  4           'TEST',
  5           SDO_KEYWORDARRAY('Transamerica Pyramid', 'San Francisco, CA'),
  6           'US',
  7           'DEFAULT'
  8        )
  9     );
 10  END;
 11  /
- ID                          0
- ADDRESSLINES
- PLACENAME                   TRANSAMERICA PYRAMID
- STREETNAME                  MONTGOMERY ST
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                  SAN FRANCISCO
- MUNICIPALITY
- REGION                      CA
- COUNTRY                     US
- POSTALCODE                  94111
- POSTALADDONCODE
- FULLPOSTALCODE              94111
- POBOX
- HOUSENUMBER                 600
- BASENAME
- STREETTYPE
- STREETTYPEBEFORE            F
- STREETTYPEATTACHED          F
- STREETPREFIX
- STREETSUFFIX
- SIDE                        R
```

```
- PERCENT                    0
- EDGEID                     23611721
- ERRORMESSAGE               ????#ENUT?B281CP?
- MATCHVECTOR                ????4101110??004?
-    # 4 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    T 1 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                   1 = Exact match
- MATCHMODE                   DEFAULT
- LONGITUDE                  -122.40305
- LATITUDE                   37.79509

PL/SQL procedure successfully completed.
```

In this example:

- the Transamerica Pyramid: 600 Montgomery Street, San Francisco, CA 94111

- **GEOCODE_ALL**

Below shows creating a stored procedure that will help in decoding the results of a call to the GEOCODE_ALL function

FORMAT_ADDR_ARRAY Procedure:

```
SQL> CREATE OR REPLACE PROCEDURE format_addr_array
  2  (
  3     address_list SDO_ADDR_ARRAY
  4  )
  5  AS
  6  BEGIN
  7     IF address_list.count() > 0 THEN
  8        FOR i IN 1..address_list.count() LOOP
  9           dbms_output.put_line ('ADDRESS['||i||']');
 10           format_geo_addr (address_list(i));
 11        END LOOP;
 12     END IF;
 13  END;
 14  /
```

```
Procedure created.
```

Below is an example of geocoding the ambiguous address "12 Presidio."

Using GEOCODE_ALL over an Ambiguous Address:

```
SQL> SET SERVEROUTPUT ON SIZE 32000
SQL> BEGIN
  2     FORMAT_ADDR_ARRAY (
  3        SDO_GCDR.GEOCODE_ALL (
  4           'TEST',
  5           SDO_KEYWORDARRAY('12 Presidio', 'San Francisco, CA
  6           'US',
  7           'DEFAULT'
  8        )
  9     );
 10  END;
 11  /
ADDRESS[1]
- ID                          1
- ADDRESSLINES
- PLACENAME
- STREETNAME                  PRESIDIO AVE
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                  SAN FRANCISCO
- MUNICIPALITY
- REGION                      CA
- COUNTRY                     US
- POSTALCODE                  94115
- POSTALADDONCODE
- FULLPOSTALCODE              94115
- POBOX
- HOUSENUMBER                 12
- BASENAME                    PRESIDIO
- STREETTYPE                  AVE
- STREETTYPEBEFORE            F
- STREETTYPEATTACHED          F
- STREETPREFIX
- STREETSUFFIX
- SIDE                        R
- PERCENT                     .8877551020408163
- EDGEID                      23614728
- ERRORMESSAGE                ????#ENU??B281CP?
- MATCHVECTOR                 ????0101410??004?
-    # 0 House or building number
-    E 1 Street prefix
```

```
-    N 0 Street base name
-    U 1 Street suffix
-    ? 4 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                2 = Street type not matched
- MATCHMODE                DEFAULT
- LONGITUDE                -122.44757091836735
- LATITUDE                 37.7915968367347
ADDRESS[2]
- ID                       1
- ADDRESSLINES
- PLACENAME
- STREETNAME               PRESIDIO BLVD
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT               SAN FRANCISCO
- MUNICIPALITY
- REGION                   CA
- COUNTRY                  US
- POSTALCODE               94129
- POSTALADDONCODE
- FULLPOSTALCODE           94129
- POBOX
- HOUSENUMBER              12
- BASENAME                 PRESIDIO
- STREETTYPE               BLVD
- STREETTYPEBEFORE         F
- STREETTYPEATTACHED       F
- STREETPREFIX
- STREETSUFFIX
- SIDE                     L
- PERCENT                  .7931034482758621
- EDGEID                   23622533
- ERRORMESSAGE             ????#ENU??B281CP?
- MATCHVECTOR              ????0101410??004?
-    # 0 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
```

```
-    ? 4 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                2 = Street type not matched
- MATCHMODE                DEFAULT
- LONGITUDE                -122.45612528011925
- LATITUDE                 37.798262171909265
ADDRESS[3]
- ID                       1
- ADDRESSLINES
- PLACENAME
- STREETNAME               PRESIDIO TER
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT               SAN FRANCISCO
- MUNICIPALITY
- REGION                   CA
- COUNTRY                  US
- POSTALCODE               94118
- POSTALADDONCODE
- FULLPOSTALCODE           94118
- POBOX
- HOUSENUMBER              12
- BASENAME                 PRESIDIO
- STREETTYPE               TER
- STREETTYPEBEFORE         F
- STREETTYPEATTACHED       F
- STREETPREFIX
- STREETSUFFIX
- SIDE                     R
- PERCENT                  .6428571428571429
- EDGEID                   28488847
- ERRORMESSAGE             ????#ENU??B281CP?
- MATCHVECTOR              ????0101410??004?
-    # 0 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    ? 4 Street type
-    ? 1 Secondary unit
```

```
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                    2 = Street type not matched
- MATCHMODE                    DEFAULT
- LONGITUDE                   -122.46105691438208
- LATITUDE                    37.788768523050976

PL/SQL procedure successfully completed.
```

Below is an example of getting the full address and geographical location of the two YMCAs in San Francisco.

Using GEOCODE_ALL over an Ambiguous Address:

```
SQL> SET SERVEROUTPUT ON SIZE 10000
SQL> BEGIN
  2     FORMAT_ADDR_ARRAY (
  3        SDO_GCDR.GEOCODE_ALL (
  4        'TEST',
  5        SDO_KEYWORDARRAY('YMCA', 'San Francisco, CA'),
  6        'US',
  7        'DEFAULT'
  8     )
  9  );
 10  END;
 11  /
ADDRESS[1]
- ID                          1
- ADDRESSLINES
- PLACENAME                   YMCA
- STREETNAME                  GOLDEN GATE AVE
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                  SAN FRANCISCO
- MUNICIPALITY
- REGION                      CA
- COUNTRY                     US
- POSTALCODE                  94102
- POSTALADDONCODE
- FULLPOSTALCODE              94102
- POBOX
- HOUSENUMBER                 220
- BASENAME
```

```
- STREETTYPE
- STREETTYPEBEFORE          F
- STREETTYPEATTACHED        F
- STREETPREFIX
- STREETSUFFIX
- SIDE                      L
- PERCENT                   0
- EDGEID                    23605184
- ERRORMESSAGE              ????#ENUT?B281CP?
- MATCHVECTOR               ????4101110??004?
-    # 4 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    T 1 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                  1 = Exact match
- MATCHMODE                  DEFAULT
- LONGITUDE                  -122.41412
- LATITUDE                   37.78184
ADDRESS[2]
- ID                         1
- ADDRESSLINES
- PLACENAME                  YMCA
- STREETNAME                 SACRAMENTO ST
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                 SAN FRANCISCO
- MUNICIPALITY
- REGION                    CA
- COUNTRY                   US
- POSTALCODE                94108
- POSTALADDONCODE
- FULLPOSTALCODE            94108
- POBOX
- HOUSENUMBER               855
- BASENAME
- STREETTYPE
- STREETTYPEBEFORE          F
```

```
- STREETTYPEATTACHED        F
- STREETPREFIX
- STREETSUFFIX
- SIDE                      R
- PERCENT                   0
- EDGEID                    23615793
- ERRORMESSAGE              ????#ENUT?B281CP?
- MATCHVECTOR               ????4101110??004?
-    # 4 House or building number
-    E 1 Street prefix
-    N 0 Street base name
-    U 1 Street suffix
-    T 1 Street type
-    ? 1 Secondary unit
-    B 0 Built-up area or city
-    1 0 Region
-    C 0 Country
-    P 4 Postal code
-    ? ? Postal add-on code
- MATCHCODE                  1 = Exact match
- MATCHMODE                  DEFAULT
- LONGITUDE                 -122.40685
- LATITUDE                  37.7932

PL/SQL procedure successfully completed.
```

In this example:

- the addresses of this business are:
  - YMCA, 220 Golden Gate Avenue
  - YMCA, 855 Sacramento Street

6.3 Geocoding Using Structured Addresses

- **GEOCODE_ADDR**

  GEOCODE_ADDR is identical to the GEOCODE function with SDO_GEO_ADDR object as input.

  Example of Calling the GEOCODE_ADDR Function:

```
SQL> SELECT SDO_GCDR.GEOCODE_ADDR
  2  (
  3    'TEST',
  4    SDO_GEO_ADDR
  5    (
  6      'US',                    -- COUNTRY
```

```
  7         'DEFAULT',                -- MATCHMODE
  8         '1200 Clay Street', -- STREET
  9         'San Francisco',       -- SETTLEMENT
 10         NULL,                      -- MUNICIPALITY
 11         'CA',                     -- REGION
 12         '94108'                   -- POSTALCODE
 13      )
 14   )
 15   FROM DUAL;

SDO_GCDR.GEOCODE_ADDR('TEST',GEO_ADDR_POI('US',--COUNTRY'MOSCONECEN
TER'-- MATCHMODE'12
--------------------------------------------------------------------------------
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), NULL, 'CLAY ST', NULL, NULL, 'SAN
FRANCISCO', NULL, 'CA', 'US', '94108', NULL, '94108', NULL, '1200', 'CLAY', 'ST', 'F', 'F',
NULL, NULL, 'L', .99, 23600695, '????#ENUT?B281CP?', 1, 'DEFAULT', -122.41274,
37.7933978, '????0101014??000?')
```

Below is an example of populating the PLACENAME attribute of the SDO_GEO_ADDR object to geocode a point without an address

A Function Producing an SDO_GEO_ADDR Object:

```
SQL> CREATE OR REPLACE FUNCTION geo_addr_poi (
  2     country    VARCHAR2,
  3     poi_name VARCHAR2
  4   )
  5   RETURN SDO_GEO_ADDR
  6   AS
  7     geo_addr SDO_GEO_ADDR := SDO_GEO_ADDR();
  8   BEGIN
  9     geo_addr.country := country;
 10     geo_addr.placename := poi_name;
 11     geo_addr.matchmode := 'DEFAULT';
 12   return geo_addr ;
 13   end;
 14   /

Function created.
```

Below is an example of using GEOCODE_ADDR function to find the location of the Moscone Center.

Example of Calling the GEOCODE_ADDR Function:

```
SQL> SELECT SDO_GCDR.GEOCODE_ADDR
  2   (
  3     'TEST',
```

```
  4     GEO_ADDR_POI
  5     (
  6       'US',                    -- COUNTRY
  7       'Moscone Center'       -- POI_NAME
  8     )
  9   )
 10   FROM DUAL;

SDO_GCDR.GEOCODE_ADDR('TEST',GEO_ADDR_POI('US',--COUNTRY'MOSCONECEN
TER'--POI_NAM
--------------------------------------------------------------------------------
SDO_GEO_ADDR(0, SDO_KEYWORDARRAY(), 'MOSCONE CENTER', 'HOWARD ST',
NULL, NULL, '
SAN FRANCISCO', NULL, 'CA', 'US', '94103', NULL, '94103', NULL, '747', NULL, NUL
L, 'F', 'F', NULL, NULL, 'R', 0, 23607005, '????#ENUT?B281CP?', 1, 'DEFAULT', -1
22.40137, 37.7841, '????4141114??404?')
```

In this example:

- the addresses of the Moscone Center: HOWARD ST, SAN FRANCISCO, 94103

6.4 Reverse Geocoding

The reverse geocoding performs the reverse operation of geocoding; given a spatial location (coordinates of a point), it returns the corresponding street address.

- **REVERSE_GEOCODE**
  Below an example of using the FORMAT_GEO_ADDR procedure to format, the result of the REVERSE_GEOCODE functions.
  Example of Calling the REVERSE_GEOCODE Function:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2    FORMAT_GEO_ADDR (
  3      SDO_GCDR.REVERSE_GEOCODE (
  4        'TEST',
  5        SDO_GEOMETRY (
  6          2001,
  7          8307,
  8          SDO_POINT_TYPE (-122.4152166, 37.7930,
  9          NULL, NULL
 10        ),
 11        'US'
 12      )
```

```
 13      );
 14   END;
 15   /
- ID                            0
- ADDRESSLINES
- PLACENAME
- STREETNAME                    CLAY ST
- INTERSECTSTREET
- SECUNIT
- SETTLEMENT                    SAN FRANCISCO
- MUNICIPALITY
- REGION                        CA
- COUNTRY                       US
- POSTALCODE                    94109
- POSTALADDONCODE
- FULLPOSTALCODE                94109
- POBOX
- HOUSENUMBER                   1351
- BASENAME                      CLAY
- STREETTYPE                    ST
- STREETTYPEBEFORE              F
- STREETTYPEATTACHED            F
- STREETPREFIX
- STREETSUFFIX
- SIDE                          R
- PERCENT                       .484531914156248
- EDGEID                        23600696
- ERRORMESSAGE
- MATCHVECTOR                   ????4141414??404?
-       4 House or building number
-       1 Street prefix
-       4 Street base name
-       1 Street suffix
-       4 Street type
-       1 Secondary unit
-       4 Built-up area or city
-       4 Region
-       0 Country
-       4 Postal code
-       ? Postal add-on code
- MATCHCODE                      1 = Exact match
- MATCHMODE                      DEFAULT
- LONGITUDE                      -122.415225677046
- LATITUDE                       37.7930717518897
```

```
PL/SQL procedure successfully completed.
```

6.5 Geocoding Business Data

- **Geocoding the Addresses: The "Naive" Approach**
  Below is an example of using the GEOCODE_AS_GEOMETRY function to update the location column.
  Populating the location Column of the branches Table:

```
SQL> UPDATE branches
  2  SET location = SDO_GCDR.GEOCODE_AS_GEOMETRY
  3  (
  4     'TEST',
  5     SDO_KEYWORDARRAY
  6     ( street_number || ' ' || street_name, city || ' ' || state || ' '
  7        || postal_code),
  8     'US'
  9  );

77 rows updated.

SQL> COMMIT;

Commit complete.
```

Populating the location Column of the branches Table for German Addresses
UPDATE branches:

```
SQL> UPDATE branches
  2  SET location = SDO_GCDR.GEOCODE_AS_GEOMETRY
  3  (
  4     'TEST',
  5     SDO_KEYWORDARRAY
  6     ( street_name || ' ' || street_number || postal_code || ' ' || city),
  7     'DE'
  8  );
SET location = SDO_GCDR.GEOCODE_AS_GEOMETRY
                        *
ERROR at line 2:
ORA-29532: Java call terminated by uncaught Java exception: Error: Cannot find
Geocoder for country:DE in user TEST
ORA-06512: at "MDSYS.SDO_GCDR", line 679
ORA-06512: at "MDSYS.SDO_GCDR", line 722
```

In this example:

- the reason of this error is the data for geocoder country DE (German) does not exist

- **Address Verification and Correction**

Below example shows geocoding the addresses in the customers table.

Address Geocoding and Correction:

```
SQL> SET SERVEROUTPUT ON SIZE 320000
SQL> DECLARE
  2     type match_counts_t is table of number;
  3
  4     input_address        sdo_geo_addr;             -- Input address to geocode
  5
  6       geo_addresses          sdo_addr_array;              -- Array of matching geocoded
addresses
  7     geo_address          sdo_geo_addr;             -- Matching address
  8     geo_location         sdo_geometry;            -- Geographical location
  9
 10     address_count        number;                   -- Addresses processed
 11     geocoded_count       number;                   -- Addresses successfully geocoded
 12     corrected_count      number;                  -- Addresses geocoded and corrected
 13      ambiguous_count      number;                       -- Ambiguous addresses (multiple
matches
 14     error_count          number;                  -- Addresses rejected
 15
 16     match_counts         match_counts_t            -- Counts per matchcode
 17        := match_counts_t();
 18
 19     update_address       boolean;                  -- Should update address ?
 20
 21   BEGIN
 22
 23     -- Clear counters
 24     address_count := 0;
 25     geocoded_count := 0;
 26     error_count := 0;
 27     corrected_count := 0;
 28     ambiguous_count := 0;
 29     match_counts.extend(20);
 30     for i in 1..match_counts.count loop
 31        match_counts(i) := 0;
 32     end loop;
 33
 34     -- Range over the customers
```

```
35    for b in
36       (select * from customersC3)
37    loop
38
39       -- Format the input address
40       input_address := sdo_geo_addr();
41       input_address.streetname    := b.street_name;
42       input_address.housenumber := b.street_number;
43       input_address.settlement    := b.city;
44       input_address.postalcode    := b.postal_code;
45       input_address.region        := b.state;
46       input_address.country       := 'US';
47       input_address.matchmode     := 'DEFAULT';
48
49       -- Geocode the address
50       geo_addresses := sdo_gcdr.geocode_addr_all (
51          'TEST',
52          input_address
53       );
54
55       -- Check results
56       update_address := false;
57       address_count := address_count + 1;
58
59       if geo_addresses.count() > 1 then
60
61          -- Address is ambiguous: reject
62          geo_location := NULL;
63          ambiguous_count := ambiguous_count + 1;
64
65       else
66          -- Extract first or only match
67          geo_address := geo_addresses(1);
68
69          -- Keep counts of matchcodes seen
70          match_counts(geo_address.matchcode) :=
71             match_counts(geo_address.matchcode) + 1;
72
73          -- The following matchcodes are accepted:
74          --     1 = exact match
75          --     2 = only street type or suffix/prefix is incorrect
76          --    10 = only postal code is incorrect
77          if geo_address.matchcode in (1,2,10) then
78             -- Geocoding succeeded: construct geometric point
```

```
79              geo_location := sdo_geometry (2001, 8307, sdo_point_type (
80                 geo_address.longitude, geo_address.latitude, null),
81                 null, null);
82              geocoded_count := geocoded_count + 1;
83
84              -- If wrong street type or postal code (matchcodes 2 or 10)
85              -- accept the geocode and correct the address in the database
86              if geo_address.matchcode <> 1 then
87                 update_address := true;
88                 corrected_count := corrected_count + 1;
89              end if;
90
91           else
92              -- For all other matchcoded, reject the geocode
93              error_count := error_count + 1;
94              geo_location := NULL;
95           end if;
96
97        end if;
98
99        -- Update location and corrected address in database
100       if update_address then
101          update customersC3
102          set location = geo_location,
103                street_name = geo_address.streetname,
104                postal_code = geo_address.postalcode
105          where id = b.id;
106       else
107          update customersC3
108          set location = geo_location
109          where id = b.id;
110       end if;
111
112    end loop;
113
114    -- Display counts of records processed
115    dbms_output.put_line ('Geocoding completed');
116    dbms_output.put_line (address_count || ' Addresses processed');
117    dbms_output.put_line (geocoded_count || ' Addresses successfully geocoded');
118    dbms_output.put_line (corrected_count || ' Addresses corrected');
119    dbms_output.put_line (ambiguous_count || ' ambiguous addresses rejected');
120    dbms_output.put_line (error_count || ' addresses with errors');
121
122    for i in 1..match_counts.count loop
```

```
123         if match_counts(i) > 0 then
124            dbms_output.put_line ('Match code '|| i || ': ' || match_counts(i));
125         end if;
126      end loop;
127
128   END;
129   /
Geocoding completed
1 Addresses processed
0 Addresses successfully geocoded
0 Addresses corrected
1 ambiguous addresses rejected
0 addresses with errors

PL/SQL procedure successfully completed.
```

- **Automatic Geocoding**

  The geocoder is invoked using simple function calls, which allows addresses to be geocoded automatically whenever an address is changed.

  Below example shows a simple trigger that automatically geocodes addresses in the branches table.

  Automatic Geocoding of the branches Table Using a Simple Trigger:

```
SQL> CREATE OR REPLACE TRIGGER branches_geocode
  2     BEFORE INSERT OR UPDATE OF street_name, street_number, postal_code, city
  3     ON branches
  4     FOR EACH ROW
  5  DECLARE
  6     geo_location SDO_GEOMETRY;
  7  BEGIN
  8     geo_location := SDO_GCDR.GEOCODE_AS_GEOMETRY (
  9        'TEST',
 10        SDO_KEYWORDARRAY (
 11           :new.street_number || ' ' || :new.street_name,
 12           :new.city || ' ' || :new.postal_code),
 13        'US'
 14     );
 15     :new.location := geo_location;
 16  END;
 17  /

Trigger created.
```

Below are examples of testing trigger when a new location is accepted.

The currently branch at 1 Van Ness Avenue:

```
SQL> SELECT name, street_number, street_name, city, postal_code, location
  2  FROM branches
  3  WHERE id = 77;

NAME        STREE_NUMBER          STREET_NAME           CITY          POSTAL_CODE
--------------------------------- ------------------------------------------------------------------------------------
LOCATION(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
BANK OF AMERICA      1            VAN NESS AVE      SAN FRANCISCO      94103

SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.41915, 37.7751038, NULL), NULL,
NULL)
```

In this example:

- the 'BANK OF AMERICA' branch is currently located in: 1, VAN NESS AVE, SAN FRANCISCO, 94103

The branch relocates to 1500 Clay Street:

```
SQL> UPDATE branches
  2  SET street_name = 'Clay Street', street_number = 1500
  3  WHERE id = 77;

1 row updated.

-- result checking

SQL> SELECT name, street_number, street_name, city, postal_code, location
  2  FROM branches
  3  WHERE id = 77;

NAME        STREE_NUMBER     STREET_NAME           CITY          POSTAL_CODE
--------------------------------- ------------------------------------------------------------------------------------
LOCATION(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
BANK OF AMERICA      1500         Clay Street       SAN FRANCISCO      94103
```

In this example:

- the branch now has the new address in 1500 Clayton Street

Below is an example of creating a trigger using the GEOCODE_ALL procedure, which can correct the address automatically in addition to simply geocoding it.

Automatic Geocoding with Address Correction:

```
SQL> CREATE OR REPLACE TRIGGER branches_geocode
  2    BEFORE INSERT OR UPDATE OF street_name, street_number, pos
  3    ON branches
  4    FOR EACH ROW
  5
  6  DECLARE
  7    input_address SDO_GEO_ADDR;
  8    geo_location SDO_GEOMETRY;
  9    geo_addresses SDO_ADDR_ARRAY;
 10    geo_address SDO_GEO_ADDR;
 11    update_address BOOLEAN;
 12
 13  BEGIN
 14    -- Format the input address
 15    input_address := sdo_geo_addr();
 16    input_address.streetname    := :new.street_name;
 17    input_address.housenumber := :new.street_number;
 18    input_address.settlement    := :new.city;
 19    input_address.postalcode    := :new.postal_code;
 20    input_address.region        := :new.state;
 21    input_address.country       := 'US';
 22    input_address.matchmode     := 'DEFAULT';
 23
 24    -- Geocode the address
 25    geo_addresses := sdo_gcdr.geocode_addr_all (
 26      'TEST',
 27      input_address
 28    );
 29
 30    -- Check results
 31    if geo_addresses.count() > 1 then
 32      -- Address is ambiguous: reject
 33      geo_location := NULL;
 34    else
 35      -- Extract first or only match
 36      geo_address := geo_addresses(1);
 37      -- The following matchcodes are accepted:
 38      --     1 = exact match
 39      --     2 = only street type or suffix/prefix is incorrect
 40      --    10 = only postal code is incorrect
 41      if geo_address.matchcode in (1,2,10) then
 42        -- Geocoding succeeded: construct geometric point
 43        geo_location := sdo_geometry (2001, 8307, sdo_point_ty
 44          geo_address.longitude, geo_address.latitude, null),
```

```
 45              null, null);
 46           -- If wrong street type or postal code (matchcodes 2 o
 47           -- accept the geocode and correct the address in the d
 48           if geo_address.matchcode <> 1 then
 49              update_address := true;
 50           end if;
 51        else
 52           -- For all other matchcodes, reject the geocode
 53           geo_location := NULL;
 54        end if;
 55     end if;
 56
 57     -- Update loaction
 58     :new.location := geo_location;
 59     -- If needed, correct address
 60     :new.street_name := geo_address.streetname;
 61     :new.postal_code := geo_address.postalcode;
 62
 63  END;
 64  /

Trigger created.
```

Below are examples of testing the trigger.

Perform the same address change of branch 77 from 1 Van Ness Avenue to 1500 Clay Street:

```
SQL> UPDATE branches
  2  SET street_name = 'Clay Street', street_number = 1500
  3  WHERE id = 77;

1 row updated.

--the result:
SQL> SELECT name, street_number, street_name, city, postal_code, location
  2  FROM branches WHERE id = 77;

NAME       STREE_NUMBER    STREET_NAME          CITY          POSTAL_CODE
------------------------------ ----------------------------------------------------------------------------
LOCATION(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
BANK OF AMERICA      1500    CLAY ST           SAN FRANCISCO       94109

SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.41768, 37.7927675, NULL), NULL,
NULL)
```

In this example:

- the geographical location is the same as computed previously
- the street name was corrected to match the name in the reference data
- the postal code is the right one for that location now

The branch moves to 1200 Montgomery Street:

```
SQL> UPDATE branches
  2  SET street_name = 'Montgommery street', street_number = 1200
  3  WHERE id = 77;

1 row updated.

--the result:
SQL> SELECT name, street_number, street_name, city, postal_code, location
  2  FROM branches WHERE id = 77;

NAME        STREE_NUMBER    STREET_NAME          CITY          POSTAL_CODE
------------------------------ ----------------------------------------------------------------------------------
LOCATION(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
BANK OF AMERICA      1200    MONTGOMERY ST        SAN FRANCISCO    94133
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.40405, 37.8001438, NULL), NULL,
NULL)
```

In this example:

- the street name was automatically corrected
- the postal code is also now correct for that section of Montgomery Street

7.0 MANIPULATING SDO_GEOMETRY

Advanced application developers often need to access and manipulate spatial objects in their applications. This section demonstrated inserting and manipulating the map geometry types into data structures in the application with basic operation such as read, decode, construct, and write geometries.

Extracting Information

An example of examining the contents of an SDO_GEOMETRY in SQL to obtain the graphical coordinates from an SDO_GEOMETRY object. This example illustrates a technique for extracting information from objects: dot notation.

Extracting Coordinates:

```
SQL> SELECT b.name,
  2             b.location.sdo_point.x b_long,
  3             b.location.sdo_point.y b_lat
  4   FROM     branches b
  5   WHERE    b.id=42 ;

NAME                                          B_LONG         B_LAT
---------------------------------- ---------- ----------
BANK OF AMERICA                              -122.4783 37.7803596
```

In this example:

- Flow scalar value are extracted from geometry objects:
  1) the geometry type (SDO_GTYPE)
  2) spatial reference system ID (SDO_SRID)
  3) the X, Y, and Z attributes of the point structure (SDO_POINT.X, .Y, and .Z)

Selecting Data Based on Spatial Relationships

An example shows the selection of customers within a quarter-mile 207 distance of a specific branch using function SDO_WITHIN_DISTANCE.

Simple Spatial Query:

```
SQL> SELECT c.name, c.phone_number
  2      FROM branches b, customers c
  3    WHERE b.id=42
  4       AND SDO_WITHIN_DISTANCE (c.location,b.location,'distance=0.25 unit=mile')
  5             = 'TRUE';

NAME                                     PHONE_NUMBER
---------------------------------- ---------------
GLOWA GARAGE                            415-7526677
PUERTOLAS PERFORMANCE                     415-7511701
CLEMENT STREET GARAGE                   415-2218868
TOPAZ HOTEL SERVICE                     415-9744400
```

ST MONICA ELEMENTARY SCHOOL

7.1 Manipulating Geometries Using PL/SQL

A sample application using SDO_GEOMETRY objects in PL/SQL to create a new branch location, computes a rectangular sales region, a delivery route for its business, and extends the delivery route as the delivery truck moves on.

Sample Application in PL/SQL:

```
SQL> DECLARE
  2     b_long                NUMBER;
  3     b_lat                 NUMBER;
  4     new_long              NUMBER;
  5     new_lat               NUMBER;
  6     new_branch_loc        SDO_GEOMETRY;
  7     sales_region          SDO_GEOMETRY;
  8     route                 SDO_GEOMETRY;
  9
 10   BEGIN
 11     -- Obtain Old location for branch id=1
 12     SELECT br.location.sdo_point.x, br.location.sdo_point.y
 13     INTO b_long, b_lat
 14     FROM branches br
 15     WHERE id=1;
 16
 17     -- Compute new coordinates: say the location is displaced by 0.0025 degrees
 18     new_long := b_long+ 0.0025;
 19     new_lat := b_lat + 0.0025;
 20
 21     -- Create new branch location using old location
 22     new_branch_loc :=
 23        point
 24        (
 25           X=>       new_long,
 26           Y=>       new_lat,
 27           SRID=> 8307
 28         ) ;
 29
 30     -- Compute sales region for this branch
 31     sales_region :=
 32        rectangle
 33        (
 34           CTR_X=> new_long,
 35           CTR_Y=> new_lat,
```

```
36            EXP_X=> 0.005,
37            EXP_Y=> 0.0025,
38            SRID=>    8307
39         ) ;
40
41      -- Create Delivery Route
42      route :=
43         line
44         (
45            FIRST_X=> -122.4804,
46            FIRST_Y=> 37.7805222,
47            NEXT_X=> -123,
48            NEXT_Y=> 38,
49            SRID=>    8307
50         ) ;
51
52      -- Update Delivery Route by adding new point
53      route :=
54         add_to_line
55         (
56            GEOM=> route,
57            POINT => POINT(-124, 39, 8307)
58         ) ;
59
60      -- Perform additional analysis such as length of route,
61      -- or # of customers in sales region (we will see examples in Chapters 8 and 9)
62      -- ...
63      -- Update geometry in branches table
64      UPDATE branches SET LOCATION = new_branch_loc WHERE id=1;
65
66   END;
67   /

PL/SQL procedure successfully completed.
```

In this example:

- three variables new_branch_loc, sales_region, and route of type SDO_GEOMETRY are declared to hold geometry objects
- use SDO_GEOMETRY objects as bind variables in static or dynamic SQL statements to updates the location of a branch
- a stored functions, point, is created to take scalar arguments and returns an SDO_GEOMETRY object
- the add_to_line function has an SDO_GEOMETRY as the first argument and returns an SDO_GEOMETRY

- **VARRAY Manipulation Primer**

  VARRAYs are short for *varying arrays*. It behaves like an array, which holds a *fixed* number of elements that can be extended and shrunk. It uses sequential numbers as subscripts, starting from 1.

  An example illustrates various array manipulations using VARRAYs.

  Manipulating VARRAYs:

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     -- Declare a type for the VARRAT
  3     TYPE MY_ARRAY_TYPE IS VARRAY(10) OF NUMBER;
  4
  5     -- Declare a array variable
  6     V                    MY_ARRAY_TYPE;
  7
  8     -- Other variables
  9     I                    NUMBER;
 10     K                    NUMBER;
 11     L                    NUMBER;
 12     ARRAY_CAPACITY   NUMBER;
 13     N_ENTRIES          NUMBER;
 14
 15   BEGIN
 16     -- Initialize the array
 17     V := MY_ARRAY_TYPE (1,2,3,4);
 18
 19     -- Get the value of a specific entry
 20     DBMS_OUTPUT.PUT_LINE('* Values for specific array entries');
 21     K := V(3);
 22     DBMS_OUTPUT.PUT_LINE('V(3)='|| V(3));
 23     I := 2;
 24     L := V(I+1);
 25     DBMS_OUTPUT.PUT_LINE('I=' || I);
 26     DBMS_OUTPUT.PUT_LINE('V(I+1)=' || V(I+1));
 27
 28     -- Find the capacity of a VARRAY:
 29     DBMS_OUTPUT.PUT_LINE('* Array capacity');
 30     ARRAY_CAPACITY := V.LIMIT();
 31     DBMS_OUTPUT.PUT_LINE('Array Capacity: V.LIMIT()='||V.LIMIT());
 32     N_ENTRIES := V.COUNT();
 33     DBMS_OUTPUT.PUT_LINE('Current Array Size: V.COUNT()='||V.COUNT
 34
 35     -- Range over all values in a VARRAY
 36     DBMS_OUTPUT.PUT_LINE('* Array Content');
 37     FOR I IN 1..V.COUNT() LOOP
```

```
38      DBMS_OUTPUT.PUT_LINE('V('||I||')=' || V(I));
39    END LOOP;
40
41    FOR I IN V.FIRST()..V.LAST() LOOP
42     DBMS_OUTPUT.PUT_LINE('V('||I||')=' || V(I));
44    END LOOP;
45
46    I := V.COUNT();
47    WHILE I IS NOT NULL LOOP
48      DBMS_OUTPUT.PUT_LINE('V('||I||')=' || V(I));
49      I := V.PRIOR(I);
50    END LOOP;
51
52    -- Extend the VARRAY
53    DBMS_OUTPUT.PUT_LINE('* Extend the array');
54    I := V.LAST();
55    V.EXTEND(2);
56    V(I+1) := 5;
57    V(I+2) := 6;
58
59    DBMS_OUTPUT.PUT_LINE('Array Capacity: V.LIMIT()='||V.LIMIT());
60    DBMS_OUTPUT.PUT_LINE('Current Array Size: V.COUNT()='||V.COUNT
61    FOR I IN 1..V.COUNT() LOOP
62      DBMS_OUTPUT.PUT_LINE('V('||I||')='|| V(I));
63    END LOOP;
64
65    -- Shrink the VARRAY
66    DBMS_OUTPUT.PUT_LINE('* Trim the array');
67    V.TRIM();
68
69    DBMS_OUTPUT.PUT_LINE('Array Capacity: V.LIMIT()='||V.LIMIT());
70    DBMS_OUTPUT.PUT_LINE('Current Array Size: V.COUNT()='||V.COUNT
71    FOR I IN 1..V.COUNT() LOOP
72      DBMS_OUTPUT.PUT_LINE('V('||I||')='|| V(I));
73    END LOOP;
74
75    -- Delete all entries from the VARRAY
76    DBMS_OUTPUT.PUT_LINE('* Empty the array');
77    V.DELETE();
78
79    DBMS_OUTPUT.PUT_LINE('Array Capacity: V.LIMIT()='||V.LIMIT());
80    DBMS_OUTPUT.PUT_LINE('Current Array Size: V.COUNT()='||V.COUNT
81    FOR I IN 1..V.COUNT() LOOP
82      DBMS_OUTPUT.PUT_LINE('V('||I||')='|| V(I));
```

```
 83       END LOOP;
 84    END;
 85    /
* Values for specific array entries
V(3)=3
I=2
V(I+1)=3
* Array Capacity: V.LIMIT()=10
Current Array Size: V.COUNT()=4
* Array Content
V(1)=1
V(2)=2
V(3)=3
V(4)=4
V(1)=1
V(2)=2
V(3)=3
V(4)=4
V(4)=4
V(3)=3
V(2)=2
V(1)=1
* Extend the array
Array Capacity: V.LIMIT()=10
Current Array Size: V.COUNT()=6
V(1)=1
V(2)=2
V(3)=3
V(4)=4
V(5)=5
V(6)=6
* Trim the array
Array Capacity: V.LIMIT()=10
Current Array Size: V.COUNT()=5
V(1)=1
V(2)=2
V(3)=3
V(4)=4
V(5)=5
* Empty the array
Array Capacity: V.LIMIT()=10
Current Array Size: V.COUNT()=0

PL/SQL procedure successfully completed.
```

- **Creating New Geometries**

  Creating new geometries using stored functions to simplify the some SQL statements and hide some of the complexities in dealing with geometries. For example, these constructors can be used to populate new branch locations or to create new sales regions.

Point Constructor

Using stored function to insert point geometries using the SDO_GEOMETRY constructor

Example of Point Constructor Function:

```
SQL> CREATE OR REPLACE FUNCTION point (
  2     x NUMBER, y NUMBER, srid NUMBER DEFAULT 8307)
  3  RETURN SDO_GEOMETRY
  4  DETERMINISTIC
  5  IS
  6  BEGIN
  7     RETURN SDO_GEOMETRY (
  8                2001, srid, SDO_POINT_TYPE (x,y,NULL), NULL, NULL);
  9  END;
 10  /

Function created.
```

In this example:

- the standard constructor of SDO_GEOMETRY are used to generate a proper point object using the arguments provided

Rectangle Constructor

Example of creating the rectangle function that is used to define a rectangular shape.

Rectangle Constructor:

```
SQL> CREATE OR REPLACE FUNCTION rectangle (
  2     ctr_x NUMBER, ctr_y NUMBER, exp_x NUMBER, exp_y NUMBER, srid NUMBER)
  3  RETURN SDO_GEOMETRY
  4  DETERMINISTIC
  5  IS
  6     r SDO_GEOMETRY;
  7  BEGIN
  8     r := SDO_GEOMETRY (
  9        2003, srid, NULL,
 10        SDO_ELEM_INFO_ARRAY (1, 1003, 3),
 11        SDO_ORDINATE_ARRAY (
 12           ctr_x - exp_x, ctr_y - exp_y,
 13           ctr_x + exp_x, ctr_y + exp_y));
 14     RETURN r;
 15  END;
```

```
 16  /

Function created.
```

Examining a search result using Rectangle function to counts the number of customers in a rectangular window, grouped by grade.

```
SQL> SELECT count(*), customer_grade
  2    FROM customers WHERE SDO_INSIDE (location,
  3    rectangle (-122.47,37.79, 0.01, 0.01, 8307)) = 'TRUE'
  4    GROUP BY customer_grade;

COUNT(*) CUSTOMER_GRADE
---------- ---------------
       307 GOLD
         4 PLATINUM
       457 SILVER

3 rows selected.
```

Line Constructor

This is an example of creating the line function to create a new line geometry with a start point and an end-point.

Line Constructor:

```
SQL> CREATE OR REPLACE FUNCTION line (
  2     first_x NUMBER, first_y NUMBER, next_x NUMBER, next_y NUMBER, srid NUMBER)
  3  RETURN SDO_GEOMETRY
  4  DETERMINISTIC
  5  IS
  6     l SDO_GEOMETRY;
  7  BEGIN
  8     l := SDO_GEOMETRY (
  9        2002, srid, NULL,
 10        SDO_ELEM_INFO_ARRAY (1, 2, 1),
 11        SDO_ORDINATE_ARRAY (
 12           first_x, first_y,
 13           next_x, next_y));
 14     RETURN l;
 15  END;
 16  /

Function created.
```

- **Extracting Information from Geometries**

  Examining the manipulation of geometries with two examples:

  - find the number of points in a geometry
  - write a function to extract a specific point from a line geometry

Counting the Number of Point in a Geometry

This is an example of creating the get_num_points function to calculate the number of points in geometry by dividing the count of elements in the SDO_ORDINATES array.

Function to Count the Number of Points in a Geometry:

```
SQL> CREATE OR REPLACE FUNCTION get_num_points (
  2     g SDO_GEOMETRY)
  3   RETURN NUMBER
  4   IS
  5   BEGIN
  6     RETURN g.SDO_ORDINATES.COUNT() / SUBSTR(g.SDO_GTYPE,1,1);
  7   END;
  8   /

Function created.
```

Examining the search result:

```
SQL> SELECT get_num_points(geom) FROM us_states WHERE state = 'California';

GET_NUM_POINTS(GEOM)
----------------------------------------
              1146

1 row selected.
```

Extracting a Point from a Line

This example of creating a function is used to extract a selected point from geometry.

Function to Extract a Point from a Geometry:

```
SQL> CREATE OR REPLACE FUNCTION get_point (
  2     geom SDO_GEOMETRY, point_number NUMBER DEFAULT 1
  3   ) RETURN SDO_GEOMETRY
  4   IS
  5     g SDO_GEOMETRY;          -- Updated Geometry
  6     d NUMBER;                -- Number of dimensions in geometry
  7     p NUMBER;                -- Index into ordinates array
  8     px NUMBER;               -- X of extracted point
  9     py NUMBER;               -- Y of extracted point
 10
 11   BEGIN
 12     -- Get the number of dimensions from the gtype
```

```
 13     d := SUBSTR (geom.SDO_GTYPE, 1, 1);
 14
 15     -- Verify that the point exists
 16     IF point_number < 1
 17     OR point_number > geom.SDO_ORDINATES.COUNT()/d THEN
 18        RETURN NULL;
 19     END IF;
 20
 21     -- Get index in ordinates array
 22     p := (point_number-1) * d + 1;
 23
 24     -- Extract the X and Y coordinates of the desired point
 25     px := geom.SDO_ORDINATES(p);
 26     py := geom.SDO_ORDINATES(p+1);
 27
 28     -- Construct and return the point
 29     RETURN
 30        SDO_GEOMETRY (
 31           2001,
 32           geom.SDO_SRID,
 33           SDO_POINT_TYPE (px, py, NULL),
 34           NULL, NULL);
 35  END;
 36  /

Function created.
```

Examples of examining this function to get the first, middle, and last points of a line string (the line that represents Interstate 95).

Getting the First,Middle, and Last Points of a Line String:

```
SQL> -- Getting the first point of a line string
SQL> SELECT get_point(geom) p
  2  FROM us_interstates
  3  WHERE interstate='I95';

P(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-80.211761, 25.74876, NULL), NULL,
NULL)

SQL> -- Getting the last point of a line string
SQL> SELECT get_point(geom, get_num_points(geom)) p
  2  FROM us_interstates
  3  WHERE interstate='I95';
```

```
P(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-74.117897, 40.75629, NULL), NULL,
NULL)

SQL> -- Getting the middle point of a line string
SQL> SELECT get_point(geom, ROUND(get_num_points(geom)/2)) p
  2  FROM us_interstates
  3  WHERE interstate='I95';

P(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-68.115753, 46.1217, NULL), NULL,
NULL)
```

- **Modifying Existing Geometries**

Remove a Point from a Line

An example of creating a remove_point function to remove points from a geometry.

Function to Remove a Point:

```
SQL> CREATE OR REPLACE FUNCTION remove_point (
  2     geom SDO_GEOMETRY, point_number NUMBER
  3  ) RETURN SDO_GEOMETRY
  4  IS
  5     g MDSYS.SDO_GEOMETRY; -- Updated Geometry
  6     d NUMBER; -- Number of dimensions in geometry
  7     p NUMBER; -- Index into ordinates array
  8     i NUMBER; -- Index into ordinates array
  9
 10  BEGIN
 11     -- Get the number of dimensions from the gtype
 12     d := SUBSTR (geom.SDO_GTYPE, 1, 1);
 13
 14     -- Get index in ordinates array
 15     -- If 0 then we want the last point
 16     IF point_number = 0 THEN
 17        p := geom.SDO_ORDINATES.COUNT() - d + 1;
 18     ELSE
 19        p := (point_number-1) * d + 1;
 20     END IF;
 21
 22     -- Verify that the point exists
 23     IF p > geom.SDO_ORDINATES.COUNT() THEN
 24        RETURN NULL;
```

```
 25     END IF;
 26
 27     -- Initialize output line with input line
 28     g := geom;
 29
 30     -- Step 1: Shift the ordinates "up"
 31     FOR i IN p..g.SDO_ORDINATES.COUNT()-d LOOP
 32        g.SDO_ORDINATES(i) := g.SDO_ORDINATES(i+d);
 33     END LOOP;
 34
 35     -- Step 2: Trim the ordinates array
 36     g.SDO_ORDINATES.TRIM (d);
 37
 38     -- Return the updated geometry
 39     RETURN g;
 40   END;
 41   /

Function created.
```

In this example:

- converting the number of the point to be removed into the index of the SDO_ORDINATE element where the ordinates of the point start (p)

Adding a Point to a Line

The add_to_line function is used to update the geometry of the point to insert and an indication of where

For example, inserting a point at the start of the line, pass the value 1. To append it at the end of the line, pass the value 0.

Function to Adding a Point in a Line String:

```
SQL> CREATE OR REPLACE FUNCTION add_to_line (
  2     geom              SDO_GEOMETRY,
  3     point             SDO_GEOMETRY,
  4     point_number NUMBER DEFAULT 0
  5   ) RETURN SDO_GEOMETRY
  6   IS
  7     g     SDO_GEOMETRY;            -- Updated geometry
  8     d     NUMBER;                  -- Number of dimensions in line geometry
  9     t     NUMBER;                  -- Geometry type
 10     p     NUMBER;                  -- Insertion point into ordinates array
 11     i     NUMBER;
 12
 13   BEGIN
 14     -- Get the number of dimensions from the gtype
 15     d := SUBSTR (geom.SDO_GTYPE, 1, 1);
```

```
16
17      -- Get index in ordinates array
18      -- If 0, then we want the last point
19      IF point_number = 0 THEN
20         p := geom.SDO_ORDINATES.COUNT() + 1;
21      ELSE
22         p := (point_number-1) * d + 1;
23      END IF;
24
25      -- Verify that the insertion point exists
26      IF point_number <> 0 THEN
27         IF p > geom.SDO_ORDINATES.LAST()
28         OR p < geom.SDO_ORDINATES.FIRST() THEN
29            RAISE_APPLICATION_ERROR (-20000, 'Invalid insertion point');
30         END IF;
31      END IF;
32
33      -- Initialize output line with input line
34      g := geom;
35
36      -- Step 1: Extend the ordinates array
37      g.SDO_ORDINATES.EXTEND(d);
38
39      -- Step 2: Shift the ordinates "down".
40      FOR i IN REVERSE p..g.SDO_ORDINATES.COUNT()-d LOOP
41         g.SDO_ORDINATES(i+d) := g.SDO_ORDINATES(i);
42      END LOOP;
43
44      -- Step 3: Store the new point
45      g.SDO_ORDINATES(p) := point.SDO_POINT.X;
46      g.SDO_ORDINATES(p+1) := point.SDO_POINT.Y;
47      IF d = 3 THEN
48         g.SDO_ORDINATES(p+2) := point.SDO_POINT.Z;
49      END IF;
50
51      -- Return the new line string
52      RETURN g;
53   END;
54   /

Function created.
```

8.0 SPATIAL INDEXES AND OPERATORS

In this section, the utilization of spatial indexes and associated operators are practiced to perform proximity analysis with spatial data. Firstly, the concepts of spatial indexes and their creation would be introduced because it ensures effective response times for queries that perform proximity analysis. Secondly, an overview of different spatial operators that Oracle Spatial supports for performing spatial analysis for indexed tables. Particularly, two type of analysis can be provided by using the operators, which are distance-based analysis (by using SDO_NN and SDO_WITHIN_DISTANCE) and interaction-based analysis (by using SDO_RELATE and SDO_FILTER). Finally, some advanced spatial indexing features that are useful for large spatial repositories are covered in this section.

8.1 Spatial Indexes

The spatial_index is internally implemented as an R-tree index or a B-tree index, which have hierarchical structure that stores rectangle approximations of geometries as key values.

- A B-tree index can specify where to place the index data.

- **Inserting Metadata for a Spatial Layer Prior to Indexing**

  Below is an example of creating the USER_SDO_GEOM_METADATA1 view, which is used to inserted spatial metadata for a spatial layer. Particularly, the spatial metadata would be identified by <table_name, column_name>. Therefore, it is important to populate the fields of this view appropriately.

  USER_SDO_GEOM_METADATA View:

```
SQL> DESCRIBE USER_SDO_GEOM_METADATA;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TABLE_NAME                                NOT NULL VARCHAR2(32)
 COLUMN_NAME                               NOT NULL VARCHAR2(1024)
 DIMINFO                                            MDSYS.SDO_DIM_ARRAY
 SRID                                               NUMBER
```

  The example below shows the code of inserting metadata for the Spatial Layer corresponding to the location column of the customers table.

  Inserting Metadata for the Spatial Layer:

```
SQL> INSERT INTO user_sdo_geom_metadata
  2   (table_name, column_name, srid, diminfo)
  3   VALUES
  4   (
  5     'CUSTOMERS', -- TABLE_NAME
```

```
 6     'LOCATION', -- COLUMN_NAME
 7     8307, -- SRID specifying a geodetic coordinate system
 8     SDO_DIM_ARRAY -- DIMINFO attribute for storing dimension bounds, tolerance
 9     (
10       SDO_DIM_ELEMENT
11       (
12         'LONGITUDE', -- DIMENSION NAME for first dimension
13         -180, -- SDO_LB for the dimension: -180 degrees
14         180, -- SDO_UB for the dimension: 180 degrees
15         0.5 -- Tolerance of 0.5 meters (not 0.5 degrees: geodetic SRID)
16       ),
17       SDO_DIM_ELEMENT
18       (
19         'LATITUDE', -- DIMENSION NAME for second dimension
20         -90, -- SDO_LB for the dimension: -90 degrees
21         90, -- SDO_UB for the dimension: 90 degrees
22         0.5 -- Tolerance of 0.5 meters (not 0.5 degrees: geodetic SRID)
23       )
24     )
25   );
```

In this example:

- the table_name and column_name fields:
  - are set to customers and location
  - identify the spatial layer
- the srid field (‘8307’): indicate a geodetic coordinate system
- the diminfo field (‘SDO_DIM_ARRAY’): specifies the bounds and tolerance for each dimension
- the first element (‘SDO_DIM_ELEMENT’) specifies:
  - the ‘Longitude’ as dimension_name
  - lower bound for this dimension is –180 degrees
  - upper bound for this dimension is 180 degrees
  - the tolerance is set to 0.5 meters
- the second element (‘SDO_DIM_ELEMENT’) specifies:
  - the ‘Longitude’ as dimension_name
  - lower bound for this dimension is –90 degrees
  - upper bound for this dimension is 90 degrees
  - the tolerance is set to 0.5 meters

- **Creating a Spatial Index**

In addition, the index has to be dropped before any recreation.

The command in below is an example of dropping a spatial index.

Dropping a Spatial Index:

```
SQL> DROP INDEX customers_sidx;
```

```
Index dropped.
```

Below is an example of re-creating a spatial index on the location column of the customers table.

Creating a Spatial Index on the location Column of the customers Table:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2   INDEXTYPE IS MDSYS.SPATIAL_INDEX;


Index created.
```

The following example identifying all spatial index tables by querying the SDO_INDEX_TABLE column in the USER_SDO_INDEX_INFO view and take appropriate steps so that they are not moved around by an unwary DBA.

Identifying the SDO_INDEX_TABLE That Stores the Spatial Index on the customers Table:

```
SQL> SELECT SDO_INDEX_TABLE FROM USER_SDO_INDEX_INFO
  2   WHERE TABLE_NAME = 'CUSTOMERS' AND COLUMN_NAME='LOCATION';


SDO_INDEX_TABLE
--------------------------------
MDRT_11270$
```

In this example:

- the result shows the name of SDO_INDEX_TABLE is called '*MDRT_11270$*'

8.2 Spatial Index Parameters

- **Parameters**

The spatial indexes can be created with different PARAMETERS, and below is the general syntax for creating a spatial index.

Syntax of Spatial Index:

```
CREATE INDEX <indexname> ON <tablename>(<columnname>)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS ('parameter_string');
```

The *parameter_string* is a list of parameter_name=value pairs. Several important parameters will be examined to used in the CREATE INDEX statement are stored in the USER_SDO_INDEX_METADATA view.

TABLESPACE Parameters

The TABLESPACE Parameter is used to specify which tablespace to use for storing the spatial index table.

Below is an example of creating an index with 'TABLESPACE=Users' which puts the spatial index table in the 'Users' tablespace.

Creating a Spatial Index in Tablespace User:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('TABLESPACE=Users');

Index created.
```

In this example:

- 'User' is the default tablespace which have been created

Below is an example of creating an index with specified INITIAL and NEXT extents in addition to the TABLESPACE.

Creating an Index with the INITIAL and NEXT Extents for an Index Table:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('TABLESPACE=users NEXT=5K INITIAL=10K');

Index created.
```

WORK_TABLESPACE Parameter

Below example shows the creation of a spatial index with a specified separate tablespace for these working tables using the WORK_TABLESPACE parameter. It would avoid fragmenting the space in a tablespace during creating and dropping tables with different sizes.

Creating an Index with WORK_TABLESPACE as User:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('WORK_TABLESPACE=SYSAUX');

Index created.

-- perform a checking for the Index with WORK_TABLESPACE
SQL> SELECT SDO_DML_BATCH_SIZE FROM USER_SDO_INDEX_METADATA
  2  WHERE SDO_INDEX_NAME = 'CUSTOMERS_SIDX';

SDO_DML_BATCH_SIZE
------------------
              1000
```

In this example:

- the 'WORK_TABLESPACE=User' places all working tables in tablespace 'User'
- this ensures the existing tablespaces holding the index and data which are not fragmented because of index creation work
- the total size (in bytes) used in this "work tablespace" will be approximately 200–300 times the number of rows in the customers table

LAYER_GTYPE Parameter

The LAYER_GTYPE parameter can help with integrity checking and speeding up the query operators. For instance, it can specify that the geometry data in the location column of the customers table are specific-type geometries such as points (by default, all types are permitted).

Below is an example of setting the parameter string to LAYER_GTYPE = POINT to indicate that the customers table has only point data.

Creating an Index for Specific-Type (Point) Geometries:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('LAYER_GTYPE=POINT');

Index created.
```

SDO_INDX_DIMS Parameter

The SDO_INDX_DIMS parameter specifies the dimensionality of the spatial_index. Below shows an example for setting the index dimensionality explicitly to 2 as default.

Creating an R-tree Index with Dimensionality Specified:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('SDO_INDX_DIMS=2');

Index created.

-- perform a checking for this Index
SQL> SELECT SDO_DML_BATCH_SIZE FROM USER_SDO_INDEX_METADATA
  2  WHERE SDO_INDEX_NAME = 'CUSTOMERS_SIDX';

SDO_DML_BATCH_SIZE
------------------
              1000
```

SDO_INDX_BRANCH_SIZE Parameter

Below is an example of creating a spatial index with SDO_INDX_BRANCH_SIZE parameter, which specifies the batch size for the batched insert/delete/update in a transaction.

Creating an Index with the SDO_DML_BATCH_SIZE Parameter:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('SDO_DML_BATCH_SIZE=1000');

Index created.
```

SDO_LEVEL Parameter

Below is an example of creating a Quadtree Type index with SDO_LEVEL parameter.

Creating a Quadtree Type of Spatial Index:

```
SQL> CREATE INDEX customers_sidx ON customers(location)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('SDO_LEVEL=8');
CREATE INDEX customers_sidx ON customers(location)
*
ERROR at line 1:
ORA-29855: error occurred in the execution of ODCIINDEXCREATE routine
ORA-13249: Geodetic indexing not supported for Quadtrees
ORA-06512: at "MDSYS.SDO_INDEX_METHOD_10I", line 10
```

In this example:

- the error shows the "Geodetic indexing not supported for Quadtrees"

- **USER_SDO_INDEX_METADATA View**

Below is an example of using the USER_SDO_INDEX_METADATAView to check the SDO_DML_BATCH_SIZE value for the index parameters.

Examining the USER_SDO_INDEX_METADATAView for Index Parameters:

```
SQL> SELECT SDO_DML_BATCH_SIZE FROM USER_SDO_INDEX_METADATA
  2  WHERE SDO_INDEX_NAME = 'CUSTOMERS_SIDX';

SDO_DML_BATCH_SIZE
------------------
              1000
```

In this example:

- the SDO_DML_BATCH_SIZE parameter is set to the default value of 1,000 which is the same to the batch size we have create

- **Spatial Index Size Requirements**

Below statement is an example of using function to estimate the size (in megabytes) of an R-tree spatial index table.

Estimating the Size of a Spatial Index on the location Column of the customers Table:

```
SQL> SELECT sdo_tune.estimate_rtree_index_size
  2  (
  3  'TEST', -- schema name
  4  'CUSTOMERS', -- table name
  5  'LOCATION' -- column name on which the spatial index is to be built
  6  ) sz
  7  FROM dual;

        SZ
----------
         1
```

8.3 Spatial Operators

In general, Oracle Spatial supports a variety of spatial operators for performing proximity analysis. Several spatial operators are tested in below with examples, such as the SDO_WITHIN_DISTANCE, the Operator SDO_NN Operator and the Operators for Spatial Interactions.

Below is the code of general syntax that is used in the spatial operators to return values.

General Syntax of Spatial Operators:

```
<spatial_operator>
(
table_geometry IN SDO_GEOMETRY (or ST_GEOMETRY),
query_geometry IN SDO_GEOMETRY (or ST_GEOMETRY)
[, parameter_string IN VARCHAR2
[, tag IN NUMBER ]]
)
='TRUE'
```

In this syntax:

- Oracle selects only those rows for which the operator evaluates to TRUE
  - where the values in table_geometry satisfy a specified operator relationship with respect to the query_geometry
- table_geometry (SDO_GEOMETRY): values
- query_geometry (SDO_GEOMETRY): location

- **SDO_WITHIN_DISTANCE Operator**

  Usually, SDO_WITHIN_DISTANCE operator is used for performing customer analysis to find all data within a specified distance from a query location.

  Two parameters, the min_resolution parameters and the max_resolution parameters, would be tested in this part, which is useful in specifying the restriction in the parameters. However, they can be used only with SDO_FILTER, SDO_WITHIN_DISTANCE, and SDO_RELATE operators, but not with the SDO_NNquery.

  Below is an example of using the SDO_WITHIN_DISTANCE operator in the WHERE clause.

  SDO_WITHIN_DISTANCE Operator Retrieving All Customers Within a Quarter-Mile Radius of a Competitor Store:

```
SQL> SELECT ct.id, ct.name
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_WITHIN_DISTANCE
  5  (ct.location, comp.location, 'DISTANCE=0.25 UNIT=MILE ' )='TRUE' ORDER BY ct.id;
```

```
        ID NAME
---------- ------------------------------------
        25 BLAKE CONSTRUCTION
        28 COLONIAL PARKING
        34 HEWLETT-PACKARD DC GOV AFFAIRS
        41 MCGREGOR PRINTING
        48 POTOMAC ELECTRIC POWER
        50 SMITH HINCHMAN AND GRYLLS
       270 METRO-FARRAGUT NORTH STATION
       271 METRO-FARRAGUT WEST STATION
       468 SAFEWAY
       809 LINCOLN SUITES
       810 HOTEL LOMBARDY
      1044 MUSEUM OF THE THIRD DIMENSION
      1526 INTERNATIONAL FINANCE
      1538 MCKENNA AND CUNEO
      2195 STEVENS ELEMENTARY SCHOOL
      6326 HOTEL LOMBARDY
      7754 EXECUTIVE INN
      7762 PHILLIPS 66
      7789 SEVEN BUILDINGS
      7821 RENAISSANCE MAYFLOWER HOTEL
      8138 ST GREGORY HOTEL
      8382 EXXON
      8792 DESTINATION HOTEL & RESORTS

23 rows selected.
```

In this example:

- there are 23 customers has been report within the distance of 0.25 mile radius of a competitor store

Using SDO_GEOM.SDO_DISTANCE function with the SDO_WITHIN_DISTANCE operator enable to report the distance of these returned customers from the corresponding store. Below is an example of this method.

SDO_WITHIN_DISTANCE Operator Retrieving All Customers in a Quarter-Mile Radius of a Competitor Store and Reporting Their Distances:

```
SQL> col dist format 999
SQL> SELECT ct.id, ct.name,
  2  SDO_GEOM.SDO_DISTANCE(ct.location, comp.location, 0.5, ' UNIT=YARD ') dist
  3  FROM competitors comp, customers ct
  4  WHERE comp.id=1
  5  AND SDO_WITHIN_DISTANCE
  6  (ct.location, comp.location, 'DISTANCE=0.25 UNIT=MILE' )='TRUE'
  7  ORDER BY ct.id;
```

```
        ID NAME                                DIST
---------- ------------------------------------ ----
        25 BLAKE CONSTRUCTION                   319
        28 COLONIAL PARKING                     398
        34 HEWLETT-PACKARD DC GOV AFFAIRS       428
        41 MCGREGOR PRINTING                    350
        48 POTOMAC ELECTRIC POWER               355
        50 SMITH HINCHMAN AND GRYLLS            252
       270 METRO-FARRAGUT NORTH STATION         345
       271 METRO-FARRAGUT WEST STATION          272
       468 SAFEWAY                              252
       809 LINCOLN SUITES                       104
       810 HOTEL LOMBARDY                       313
      1044 MUSEUM OF THE THIRD DIMENSION        153
      1526 INTERNATIONAL FINANCE                236
      1538 MCKENNA AND CUNEO                     97
      2195 STEVENS ELEMENTARY SCHOOL            305
      6326 HOTEL LOMBARDY                       329
      7754 EXECUTIVE INN                        375
      7762 PHILLIPS 66                          303
      7789 SEVEN BUILDINGS                      355
      7821 RENAISSANCE MAYFLOWER HOTEL          322
      8138 ST GREGORY HOTEL                     359
      8382 EXXON                                326
      8792 DESTINATION HOTEL & RESORTS          159

23 rows selected.
```

In this example:

- the results show 23 customers with the their distance from the corresponding store within the distance of 0.5 yard radius

When displaying the customers for the specified store on a map, several additional background layers would need to be retrieved to construct the map such as streets stored in the map_streets. Below is an example of shows the SQL using SDO_WITHIN_DISTANCE operator to get the street names within the 0.25-mile radius of a competitor store.

Getting the Streets around 0.25 Miles of the Competitor Store:

```
SQL> SELECT s.street_name
  2  FROM competitors comp, map_streets s
  3  WHERE comp.id=1
  4  AND SDO_WITHIN_DISTANCE
  5  (s.geometry, comp.location,
  6  'DISTANCE=0.25 UNIT=MILE ' )='TRUE'
```

```
  7   ORDER BY s.street_name;

STREET_NAME
--------------------------------------------------------------------------------
18TH ST NW
19TH ST NW
20TH ST NW
21ST ST NW
CONSTITUTION CT NW
DESALES ST NW
EYE ST NW
H ST NW
JEFFERSON PL NW
L ST NW

10 rows selected.
```

Min_resolution Parameters

Below is a modified example using min_resolution parameter to obtain only those streets that are at least 200 meters in length.

Getting the Streets around 0.25 Miles of the Competitor Store:

```
SQL> SELECT s.street_name
  2   FROM competitors comp, map_streets s
  3   WHERE comp.id=1
  4   AND SDO_WITHIN_DISTANCE
  5   (s.geometry, comp.location,
  6   'DISTANCE=0.25 UNIT=MILE min_resolution=200 ' )='TRUE'
  7   ORDER BY s.street_name;

STREET_NAME
--------------------------------------------------------------------------------
18TH ST NW
19TH ST NW
20TH ST NW
21ST ST NW
EYE ST NW
H ST NW
L ST NW

7 rows selected.
```

Max_resolution Parameters

Below is an example using max_resolution parameter to obtain only those streets that are at least 500 meters in length.

Getting the Streets around 0.25 Miles of the Competitor Store:

```
SQL> SELECT s.street_name
  2  FROM competitors comp, map_streets s
  3  WHERE comp.id=1
  4  AND SDO_WITHIN_DISTANCE
  5  (s.geometry, comp.location,
  6  'DISTANCE=0.25 UNIT=MILE min_resolution=200 max_resolution=500 ' )='TRUE'
  7  ORDER BY s.street_name;

STREET_NAME
--------------------------------------------------------------------------------
21ST ST NW
H ST NW

2 rows selected.
```

- **SDO_NN Operator**

SDO_NN operator is also called nearest-neighbor operator, which is useful in performing customer analysis to find the nearest neighbors to a query location. In addition, it facilitates proximity analysis in the business application.

Below is an example of using SDO_NN operator to identify the nearest customers to a competitor store (whose ID is 1).

A Simple Example of the SDO_NN Operator:

```
SQL> SELECT ct.id, ct.name
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_NN(ct.location, comp.location)='TRUE' ;

        ID NAME
---------- ------------------------------------
      1538 MCKENNA AND CUNEO
     .....
      7937 DOCKSIDE BOAT & BED

3195 rows selected.
```

In general, the returns of selection can be limited by specifying ROWNUM<=N in the preceding SQL (N is the number of neighbors that we are interested in). Below is an example of retrieving the five nearest customers to the competitor id=1 with N is 5.

SDO_NN Operator Retrieving the Five Nearest Customers to a Specific Competitor:

```
SQL> SELECT ct.id, ct.name, ct.customer_grade
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_NN(ct.location, comp.location)='TRUE'
  5  AND ROWNUM<=5
  6  ORDER BY ct.id;

        ID NAME                                CUSTOMER_GRADE
---------- ------------------------------------ ---------------
       809 LINCOLN SUITES                       GOLD
      1044 MUSEUM OF THE THIRD DIMENSION        SILVER
      1526 INTERNATIONAL FINANCE                SILVER
      1538 MCKENNA AND CUNEO                    SILVER
      8792 DESTINATION HOTEL & RESORTS          GOLD

5 rows selected.
```

The customers have been graded into GOLD, SILVER, PLATINUM, and other categories.

The example in below returns the five nearest GOLD customers instead of any nearest five customers for competitor id=1 using customer_grade='GOLD' in WHERE clause.

SDO_NN Operator Retrieving the Five GOLD Customers Nearest to a Specific Competitor:

```
SQL> SELECT ct.id, ct.name, ct.customer_grade
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND ct.customer_grade='GOLD'
  5  AND SDO_NN(ct.location, comp.location)='TRUE'
  6  AND ROWNUM<=5
  7  ORDER BY ct.id;

        ID NAME                                CUSTOMER_GRADE
---------- ------------------------------------ ---------------
       809 LINCOLN SUITES                       GOLD
       810 HOTEL LOMBARDY                       GOLD
      6326 HOTEL LOMBARDY                       GOLD
      7821 RENAISSANCE MAYFLOWER HOTEL          GOLD
      8792 DESTINATION HOTEL & RESORTS          GOLD

5 rows selected.
```

SDO_BATCH_SIZE Tuning Parameter

SDO_BATCH_SIZE parameter enable to set the "batch" size which is determined by the index. Below example shows the SQL with the SDO_BATCH_SIZE parameter specified as 100.

SDO_NN Operator Retrieving the Five GOLD Customers Nearest to a Competitor:

```
SQL> SELECT ct.id, ct.name, ct.customer_grade
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND ct.customer_grade='GOLD'
  5  AND SDO_NN(ct.location, comp.location, 'SDO_BATCH_SIZE=100' )='TRUE'
  6  AND ROWNUM<=5
  7  ORDER BY ct.id;

        ID NAME                                CUSTOMER_GRADE
---------- ------------------------------------ ---------------
       809 LINCOLN SUITES                       GOLD
       810 HOTEL LOMBARDY                       GOLD
      6326 HOTEL LOMBARDY                       GOLD
      7821 RENAISSANCE MAYFLOWER HOTEL          GOLD
      8792 DESTINATION HOTEL & RESORTS          GOLD

5 rows selected.
```

SDO_NUM_RES Tuning Parameter

The SDO_NUM_RES=<N> parameter enable the SDO_NN operator returns exactly N neighbors in a faster way.

Below is an example of using this parameter to retrieve the five GOLD customers, which are nearest to a competitor.

SDO_NN Operator Retrieving the Five Customers Nearest to a Specific Competitor:

```
SQL> SELECT ct.id, ct.name, ct.customer_grade
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_NN(ct.location, comp.location, 'SDO_NUM_RES=5')='TRUE' ;

        ID NAME                                CUSTOMER_GRADE
---------- ------------------------------------ ---------------
       809 LINCOLN SUITES                       GOLD
      1044 MUSEUM OF THE THIRD DIMENSION        SILVER
      1526 INTERNATIONAL FINANCE                SILVER
      1538 MCKENNA AND CUNEO                    SILVER
      8792 DESTINATION HOTEL & RESORTS          GOLD

5 rows selected.
```

In this example:

- the parameter *SDO_NUM_RES=5* in the SDO_NN invocation
- the spatial index retrieves exactly five neighbors

SDO_NN with the SDO_NN_DISTANCE Operator

SDO_NN_DISTANCE operator is specified as part of the SELECT list and is bound to an SDO_NN operator in the WHERE clause. It can augment the SDO_NN operator with an ancillary operator to provide the distance of each neighbor.

Below are the examples of SQL with SDO_NN_DISTANCE operator to retrieve the distances of the neighbors with different parameters and situations.

SDO_NUM_RES Tuning Parameter

SDO_NN Operator Retrieving the Five GOLD Customers Nearest to a Specific Competitor Along with Their Distances:

```
SQL> col dist format 999
SQL> SELECT ct.id, ct.name, ct.customer_grade, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_NN(ct.location, comp.location, 'SDO_NUM_RES=5',1)='TRUE'
  5 ORDER BY ct.id;

        ID NAME                                CUSTOMER_GRADE    DIST
---------- ----------------------------------- --------------- ----
       809 LINCOLN SUITES                      GOLD               95
       810 HOTEL LOMBARDY                      GOLD              286
      6326 HOTEL LOMBARDY                      GOLD              301
      7821 RENAISSANCE MAYFLOWER HOTEL         GOLD              295
      8792 DESTINATION HOTEL & RESORTS         GOLD              146

5 rows selected.
```

Rewriting previous example with Mile as the Distance Unit:

```
SQL> col dist format 9.99
SQL> SELECT ct.id, ct.name, ct.customer_grade, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_NN(ct.location, comp.location, 'SDO_NUM_RES=5 UNIT=MILE',1)='TRUE'
  5  ORDER BY ct.id;

        ID NAME                                CUSTOMER_GRADE     DIST
---------- ----------------------------------- --------------- -----
       809 LINCOLN SUITES                      GOLD              .06
      1044 MUSEUM OF THE THIRD DIMENSION       SILVER            .09
      1526 INTERNATIONAL FINANCE               SILVER            .13
```

```
      1538 MCKENNA AND CUNEO                    SILVER          .06
      8792 DESTINATION HOTEL & RESORTS          GOLD            .09


5 rows selected.
```

Limit the Search Space to 0.1 Mile Distance:

```
SQL> col dist format 9.99
SQL> SELECT ct.id, ct.name, ct.customer_grade, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4     AND SDO_NN(ct.location, comp.location, 'SDO_NUM_RES=5 DISTANCE=0.1
UNIT=MILE',1)='TRUE'
  5  ORDER BY ct.id;

        ID NAME                                  CUSTOMER_GRADE     DIST
---------- ------------------------------------- --------------- -----
       809 LINCOLN SUITES                         GOLD              .06
      1044 MUSEUM OF THE THIRD DIMENSION          SILVER            .09
      1538 MCKENNA AND CUNEO                      SILVER            .06
      8792 DESTINATION HOTEL & RESORTS            GOLD              .09

4 rows selected.
```

SDO_BATCH_SIZE Tuning Parameter

SDO_NN Operator Retrieving the Five GOLD Customers Nearest to a Specific Competitor Along with Their Distances:

```
SQL> SQL> SELECT ct.id, ct.name, ct.customer_grade, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_NN(ct.location, comp.location, 'SDO_BATCH_SIZE=100', 1 )='TRUE'
  5  ORDER BY ct.id;

        ID NAME                                  CUSTOMER_GRADE     DIST
---------- ------------------------------------- --------------- -----
       809 LINCOLN SUITES                         GOLD              .06
      1044 MUSEUM OF THE THIRD DIMENSION          SILVER            .09
      1526 INTERNATIONAL FINANCE                  SILVER            .13
      1538 MCKENNA AND CUNEO                      SILVER            .06
      8792 DESTINATION HOTEL & RESORTS            GOLD              .09

5 rows selected.
```

Rewriting previous example with Mile as the Distance Unit:

```
SQL> col dist format 9.99
SQL> SELECT ct.id, ct.name, ct.customer_grade, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND ct.customer_grade='GOLD'
  5  AND SDO_NN
  6  (ct.location, comp.location, 'SDO_BATCH_SIZE=100 UNIT=MILE', 1 )='TRUE'
  7  AND ROWNUM<=5
  8  ORDER BY ct.id;

        ID NAME                                CUSTOMER_GRADE     DIST
---------- ----------------------------------- --------------- -----
         809 LINCOLN SUITES                     GOLD              .06
        1044 MUSEUM OF THE THIRD DIMENSION      SILVER            .09
        1526 INTERNATIONAL FINANCE              SILVER            .13
        1538 MCKENNA AND CUNEO                  SILVER            .06
        8792 DESTINATION HOTEL & RESORTS        GOLD              .09

5 rows selected.
```

Limit the Distance to 0.1 Mile:

```
SQL> col dist format 9.99
SQL> SELECT ct.id, ct.name, ct.customer_grade, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND ct.customer_grade='GOLD'
  5  AND SDO_NN
  6   (ct.location, comp.location, 'SDO_BATCH_SIZE=100 DISTANCE=0.1 UNIT=MILE',
1 )='TRUE'
  7  AND ROWNUM<=5
  8  ORDER BY ct.id;

          ID NAME                               CUSTOMER_GRADE     DIST
---------- ----------------------------------- --------------- -----
         809 LINCOLN SUITES                     GOLD              .06
        8792 DESTINATION HOTEL & RESORTS        GOLD              .09

2 rows selected.
```

- **Operators for Spatial Interactions (Relationships)**

  The operators in this part are used to find locations/geometries that interact with query geometries. Indeed, these operators are used frequently in applications for analysis using buffer zones.

Below are the examples of using the SDO_GEOM.SDO_BUFFER function to construct quarter-mile buffer zones around competitor

Creating the Sales Region (Area of Influence) for Each Competitor/Branch:

```
-- create sales region table for competitor
SQL> CREATE TABLE COMPETITORS_SALES_REGIONS AS
  2  SELECT id, name, SDO_GEOM.SDO_BUFFER
  3  (cmp.location, 0.25, 0.5, 'UNIT=MILE ARC_TOLERANCE=0.005') geom
  4  FROM competitors cmp;

Table created.

-- create sales region table for Branch
SQL> CREATE TABLE SALES_REGION AS
  2  SELECT id, name, SDO_GEOM.SDO_BUFFER
  3  (b.location, 0.25, 0.5, 'UNIT=MILE ARC_TOLERANCE=0.005') geom
  4  FROM branches b;

Table created.
```

Creating Indexes on Sales Regions of Competitors/Branches:

```
-- Metadata for Sales_regions table: copied from the metadata for Branches table
SQL> INSERT INTO USER_SDO_GEOM_METADATA
  2  SELECT 'SALES_REGION','GEOM', DIMINFO, SRID
  3  FROM USER_SDO_GEOM_METADATA
  4  WHERE TABLE_NAME='BRANCHES';

1 row created.

-- Metadata for Competitors_regions table: -- copied from the metadata for Branches table
SQL> INSERT INTO USER_SDO_GEOM_METADATA
  2  SELECT 'COMPETITORS_SALES_REGIONS','GEOM', DIMINFO, SRID
  3  FROM USER_SDO_GEOM_METADATA
  4  WHERE TABLE_NAME='COMPETITORS';

1 row created.

-- Index-creation for Sales_regions table
SQL> CREATE INDEX sr_sidx ON sales_region(geom)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX;

Index created.

-- Index-creation for Competitors_sales_regions table
```

```
SQL> CREATE INDEX cr_sidx ON competitors_sales_regions(geom)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX;

Index created.
```

SDO_FILTER Operator

The SDO_FILTER operator is an approximation of other interaction-based operators, which always returns a superset of results for other interaction-based operators. It classifies all rows of a table where the MBRs of the column geometry intersect with the MBR of a specified query geometry.

Below is an example of using the SDO_FILTER operator to identify all customers within a competitor's area of influence.

SDO_FILTER Operator Retrieving All Customers within a Competitor Service Area:

```
SQL> SELECT ct.id, ct.name
  2  FROM competitors_sales_regions comp, customers
  3  WHERE comp.id=1
  4  AND SDO_FILTER(ct.location, comp.geom)='TRUE'
  5  ORDER BY ct.id;

        ID NAME
---------- ------------------------------------
        25 BLAKE CONSTRUCTION
        28 COLONIAL PARKING
        34 HEWLETT-PACKARD DC GOV AFFAIRS
        38 KIPLINGER WASHINGTON EDITORS
        41 MCGREGOR PRINTING
        42 MCI COMMUNICATIONS
        48 POTOMAC ELECTRIC POWER
        50 SMITH HINCHMAN AND GRYLLS
       270 METRO-FARRAGUT NORTH STATION
       271 METRO-FARRAGUT WEST STATION
       468 SAFEWAY
       809 LINCOLN SUITES
       810 HOTEL LOMBARDY
      1044 MUSEUM OF THE THIRD DIMENSION
      1081 GEORGE WASHINGTON UNIVERSITY
      1178 AVIS RENT-A-CAR
      1526 INTERNATIONAL FINANCE
      1538 MCKENNA AND CUNEO
      1901 CLUB QUARTERS WASHINGTON
      2195 STEVENS ELEMENTARY SCHOOL
      6326 HOTEL LOMBARDY
      7387 ELLIPSE
      7754 EXECUTIVE INN
```

```
        7762 PHILLIPS 66
        7789 SEVEN BUILDINGS
        7821 RENAISSANCE MAYFLOWER HOTEL
        8138 ST GREGORY HOTEL
        8382 EXXON
        8792 DESTINATION HOTEL & RESORTS
        8793 LOEWS HOTELS REGIONAL

30 rows selected.
```

Below example is a typical query from Oracle MapViewer using the SDO_FILTER operator to render maps.

Typical Query from MapViewer Using the SDO_FILTER Operator:

```
SQL> SELECT location
  2  FROM customers
  3  WHERE SDO_FILTER
  4  (
  5     location,
  6     SDO_GEOMETRY
  7     (
  8        2003, 8307, null,
  9        SDO_ELEM_INFO_ARRAY(1, 1003, 3), -- Rectangle query window
 10        SDO_ORDINATE_ARRAY(-122.43886,37.78284,-122.427195,37.79284)
 11     )
 12  ) = 'TRUE';

LOCATION(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.42997, 37.786498, NULL), NULL,
NULL)
……
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-122.43063, 37.7873349, NULL), NULL,
NULL)

40 rows selected.
```

SDO_RELATR Operator

The SDO_RELATE operator provides this function to find geometries that interact with the query in a specified manner in the customer analysis.

Below is an example identifies all customers inside or on the boundary of the buffer zones of each competitor store using the ANYINTERACT interaction mask.

SDO_RELATE Operator Retrieving All Customers in a Quarter-Mile Buffer Zone of a Competitor Store with a specific competitor (id=1):

```
SQL> SELECT ct.id, ct.name
  2  FROM competitors_sales_regions comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_RELATE(ct.location, comp.geom, 'MASK=ANYINTERACT ' )='TRUE'
  5  ORDER BY ct.id;

        ID NAME
---------- ------------------------------------
        25 BLAKE CONSTRUCTION
        28 COLONIAL PARKING
        34 HEWLETT-PACKARD DC GOV AFFAIRS
        41 MCGREGOR PRINTING
        48 POTOMAC ELECTRIC POWER
        50 SMITH HINCHMAN AND GRYLLS
       270 METRO-FARRAGUT NORTH STATION
       271 METRO-FARRAGUT WEST STATION
       468 SAFEWAY
       809 LINCOLN SUITES
       810 HOTEL LOMBARDY
      1044 MUSEUM OF THE THIRD DIMENSION
      1526 INTERNATIONAL FINANCE
      1538 MCKENNA AND CUNEO
      2195 STEVENS ELEMENTARY SCHOOL
      6326 HOTEL LOMBARDY
      7754 EXECUTIVE INN
      7762 PHILLIPS 66
      7789 SEVEN BUILDINGS
      7821 RENAISSANCE MAYFLOWER HOTEL
      8138 ST GREGORY HOTEL
      8382 EXXON
      8792 DESTINATION HOTEL & RESORTS

23 rows selected.
```

To identify the DISJOINT relationship，the SQL in below identifies all customers that are disjoint from competitor region (id=1), which use negation of ANYINTERACT.

Identifying a DISJOINT Relationship:

```
-- example of DISJOINT Relationship
SQL> SELECT ct1.id, ct1.name
  2  FROM customers ct1
  3  WHERE NOT EXISTS
  4  (
  5     SELECT 'X'
  6     FROM competitors_sales_regions comp, customers ct2
```

```
  7     WHERE comp.id=1
  8     AND ct1.id = ct2.id
  9     AND SDO_RELATE(ct2.location, comp.geom, 'MASK=ANYINTERACT')='TRUE'
 10  );

no rows selected.
```

-- example of JOINT Relationship

```
SQL> SELECT ct1.id, ct1.name
  2  FROM customers ct1
  3  WHERE EXISTS
  4  (
  5     SELECT 'X'
  6     FROM competitors_sales_regions comp, customers ct2
  7     WHERE comp.id=1
  8     AND ct1.id = ct2.id
  9     AND SDO_RELATE(ct2.location, comp.geom, 'MASK=ANYINTERACT')='TRUE'
 10  );

        ID NAME
---------- ----------------------------------
        25 BLAKE CONSTRUCTION
        28 COLONIAL PARKING
        34 HEWLETT-PACKARD DC GOV AFFAIRS
        41 MCGREGOR PRINTING
       809 LINCOLN SUITES
       810 HOTEL LOMBARDY
        48 POTOMAC ELECTRIC POWER
        50 SMITH HINCHMAN AND GRYLLS
       270 METRO-FARRAGUT NORTH STATION
       271 METRO-FARRAGUT WEST STATION
       468 SAFEWAY
      1044 MUSEUM OF THE THIRD DIMENSION
      1526 INTERNATIONAL FINANCE
      1538 MCKENNA AND CUNEO
      2195 STEVENS ELEMENTARY SCHOOL
      7754 EXECUTIVE INN
      7762 PHILLIPS 66
      6326 HOTEL LOMBARDY
      8792 DESTINATION HOTEL & RESORTS
      7789 SEVEN BUILDINGS
      7821 RENAISSANCE MAYFLOWER HOTEL
      8138 ST GREGORY HOTEL
      8382 EXXON
```

```
23 rows selected.
```

Below are examples that analyze the influence of the competitors inside and outside the District of Columbia (D.C.) region using specifying multiple masks with different relationships.

ANYINTERACT Relationship

SDO_RELATE Operator Identifying All Competitors in the D.C. Region:

```
SQL> SELECT COUNT(*)
  2  FROM us_states st, competitors_sales_regions comp
  3  WHERE st.state_abrv='DC'
  4  AND SDO_RELATE(comp.geom, st.geom, 'MASK=ANYINTERACT' )='TRUE';

  COUNT(*)
----------
       286
```

INSIDE Relationship

SDO_RELATE Operator Identifying All Competitors in the D.C. Region:

```
SQL> SELECT COUNT(*)
  2  FROM us_states st, competitors_sales_regions comp
  3  WHERE st.state_abrv='DC'
  4  AND SDO_RELATE(comp.geom, st.geom, 'MASK=INSIDE' )='TRUE';

  COUNT(*)
----------
       268
```

OVERLAPBDYINTERSECT Relationship

SDO_RELATE Operator Identifying All Competitors That Overlap the D.C. Region:

```
SQL> SELECT COUNT(*)
  2  FROM us_states st, competitors_sales_regions comp
  3  WHERE st.state_abrv='DC'
  4     AND SDO_RELATE(comp.geom, st.geom, 'MASK=OVERLAPBDYINTERSECT
' )='TRUE' ;

  COUNT(*)
----------
        18
```

ANYINTERACT Relationship

Identifying Sales Regions That Intersect a Specific Sales Region (id=51):

```
SQL> SELECT sr1.id
```

```
  2  FROM sales_region sr2, sales_region sr1
  3  WHERE sr2.id=51
  4  AND sr1.id <> 51
  5  AND SDO_RELATE(sr1.geom, sr2.geom, 'MASK=ANYINTERACT' )='TRUE';

        ID
----------
        63
        54
        72
        69
        43
        66
        50
        75
        76

9 rows selected.
```

Multiple Maskes in SDO_RELATE

In addition, a plus sign (+) is used in the mask specification to combine multiple masks. Below is an example of using multiple masks to indentify the number of sales regions overlap the query sales region (id=51), which involves specifying two masks, OVERLAPBDYDISJOINT and OVERLAPBDYINTERSECT, in the parameter_string of the SDO_RELATE operator.

Identifying All Sales Regions That Overlap a Specific Sales Region (id=51):

```
SQL> SELECT sr1.id
  2  FROM sales_region sr2, sales_region sr1
  3  WHERE sr2.id=51
  4  AND sr1.id <> 51
  5  AND SDO_RELATE
  6                    (sr1.geom,                    sr2.geom,
'MASK=OVERLAPBDYDISJOINT+OVERLAPBDYINTERSECT')='TRUE' ;

        ID
----------
        63
        54
        72
        69
        43
        66
        50
        75
```

```
8 rows selected.
```

Below is an example of the query to search the sales region (id=51), which is "touching" it.

Verifying That a Sales Region Touches another Sales Region (id=51):

```
SQL> SELECT a.id
  2  FROM sales_region b, sales_region a
  3  WHERE b.id=51
  4  AND a.id <> 51
  5  AND SDO_RELATE(a.geom, b.geom, 'MASK=TOUCH' )='TRUE' ;

        ID
----------
        76
1 row selected.
```

In the two previous examples:

- the total number of Sales Regions that intersect a specific sales region (id=51) is nine
- eight are overlapping
- one is touching the border of the D.C. boundary

Tuning Parameter for SDO_RELATE

Below is an example of adding the tuning parameter SDO_LEVEL to an SDO_RELATE operator.

Adding the SDO_LEVEL=6 Parameter to an SDO_RELATE Query:

```
SQL> SELECT COUNT(*)
  2  FROM us_states st, competitors_sales_regions comp
  3  WHERE st.state_abrv='DC'
  4  AND SDO_RELATE(comp.geom, st.geom, 'MASK=INSIDE SDO_LEVEL=6' )='TRUE' ;

  COUNT(*)
----------
       268
```

- **Hints for Spatial Operators**

In general, the execution plan for the SQL statement involving the operator can determine whether a spatial operator is evaluated using the spatial index. To execute this, the utlxplan script has to be loaded first and then use SET AUTOTRACE ON to view the execution plan output.

Below are the examples of using the EXPLAIN PLAN statement.

Explaining the Execution Plan for a SQL Statement:

```
SQL> @?/rdbms/admin/utlxplan -- Load only once
SP2-0310: unable to open file "C:\DevSuiteHome_1/rdbms/admin/utlxplan.sql"
```

```
SQL> SET AUTOTRACE ON
SQL> SELECT ct.id
  2  FROM customers ct
  3  WHERE SDO_WITHIN_DISTANCE
  4  (
  5     ct.location,
  6     ( SELECT location FROM competitors WHERE id=1),
  7     'DISTANCE=0.25 UNIT=MILE '
  8  )='TRUE'
  9  ;

        ID
----------
        25
      7821
      8138
        28
        41
      2195
       468
       810
      8792
      1044
      6326
      8382
      1538
      7789
        48
       809
      1526
        50
       271
      7762
       270
      7754
        34

23 rows selected.

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=5 Card=32 Bytes=48
          64)
```

```
   1      0    TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS' (TABLE) (Cost
             =3 Card=32 Bytes=4864)

   2      1       DOMAIN INDEX OF 'CUSTOMERS_SIDX' (INDEX (DOMAIN))
   3      2          TABLE ACCESS (BY INDEX ROWID) OF 'COMPETITORS' (TABLE)
              (Cost=2 Card=1 Bytes=137)

   4      3             INDEX (UNIQUE SCAN) OF 'COMPETITORS_PK' (INDEX (UNIQ
             UE)) (Cost=1 Card=1)

Statistics
----------------------------------------------------------
          219  recursive calls
            2  db block gets
         2855  consistent gets
            0  physical reads
            0  redo size
          738  bytes sent via SQL*Net to client
          519  bytes received via SQL*Net from client
            3  SQL*Net roundtrips to/from client
            2  sorts (memory)
            0  sorts (disk)
           23  rows processed
```

SDO_NN Operator Retrieving the Five GOLD Customers Nearest to a Specific Competitor -- Along with Their Distances:

```
SQL> col dist format 999
SQL> SELECT ct.id, ct.customer_grade, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND ct.customer_grade='GOLD'
  5  AND SDO_NN(ct.location, comp.location, 'SDO_BATCH_SIZE=100', 1 )='TRUE'
  6  AND ROWNUM<=5
  7  ORDER BY ct.id;

        ID CUSTOMER_GRADE    DIST
---------- --------------- ----
       809 GOLD                95
       810 GOLD               286
      6326 GOLD               301
      7821 GOLD               295
      8792 GOLD               146

Execution Plan
```

```
------------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=12 Card=5 Bytes=14
          45)

   1    0   SORT (ORDER BY) (Cost=12 Card=5 Bytes=1445)
   2    1     COUNT (STOPKEY)
   3    2       NESTED LOOPS (Cost=11 Card=11 Bytes=3179)
   4    3         TABLE ACCESS (BY INDEX ROWID) OF 'COMPETITORS' (TABL
          E) (Cost=2 Card=1 Bytes=137)

   5    4           INDEX (UNIQUE SCAN) OF 'COMPETITORS_PK' (INDEX (UN
          IQUE)) (Cost=1 Card=1)

   6    3         TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS' (TABLE)
           (Cost=11 Card=11 Bytes=1672)

   7    6           DOMAIN INDEX OF 'CUSTOMERS_SIDX' (INDEX (DOMAIN))

Statistics
------------------------------------------------------------
         437  recursive calls
        3887  db block gets
        2539  consistent gets
           1  physical reads
           0  redo size
         657  bytes sent via SQL*Net to client
         508  bytes received via SQL*Net from client
           2  SQL*Net roundtrips to/from client
           3  sorts (memory)
           0  sorts (disk)
           5  rows processed
```

Creating an Index on customer_grade to rerun the previous example:

```
SQL> CREATE INDEX cust_grade ON CUSTOMERS(CUSTOMER_GRADE);

Index created.

SQL> col dist format 9999
SQL> SELECT ct.id, ct.CUSTOMER_GRADE, SDO_NN_DISTANCE(1) dist
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND ct.customer_grade='GOLD'
  5  AND SDO_NN(ct.location, comp.location, 'SDO_BATCH_SIZE=100', 1 )='TRUE'
  6  AND ROWNUM<=5
```

```
  7   ORDER BY ct.id;
SELECT ct.id, ct.customer_grade, SDO_NN_DISTANCE(1) dist
*
ERROR at line 1:
ORA-13249: SDO_NN cannot be evaluated without using index
ORA-06512: at "MDSYS.MD", line 1723
ORA-06512: at "MDSYS.MDERR", line 17
ORA-06512: at "MDSYS.PRVT_IDX", line 9
```

In this example:

- there is a error when using the index, so the selection can not be generated

Usage of Hints with SDO_NN and Other Operators on the Same Table:

```
SQL> SELECT /*+ NO_INDEX(ct cust_grade) INDEX(ct customers_sidx) */
  2   ct.id, ct.customer_grade, SDO_NN_DISTANCE(1) dist FROM
  3   competitors comp, customers ct
  4   WHERE comp.id=1
  5   AND ct.customer_grade='GOLD'
  6   AND SDO_NN(ct.location, comp.location, 'SDO_BATCH_SIZE=100', 1 )='TRUE'
  7   AND ROWNUM<=5
  8   ORDER BY ct.id;

        ID CUSTOMER_GRADE     DIST
---------- --------------- -----
       809 GOLD                95
       810 GOLD               286
      6326 GOLD               301
      7821 GOLD               295
      8792 GOLD               146

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=12 Card=5 Bytes=14
          45)

   1    0   SORT (ORDER BY) (Cost=12 Card=5 Bytes=1445)
   2    1     COUNT (STOPKEY)
   3    2       NESTED LOOPS (Cost=11 Card=11 Bytes=3179)
   4    3         TABLE ACCESS (BY INDEX ROWID) OF 'COMPETITORS' (TABL
          E) (Cost=2 Card=1 Bytes=137)

   5    4           INDEX (UNIQUE SCAN) OF 'COMPETITORS_PK' (INDEX (UN
          IQUE)) (Cost=1 Card=1)

   6    3         TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS' (TABLE)
```

```
              (Cost=11 Card=11 Bytes=1672)

   7     6              DOMAIN INDEX OF 'CUSTOMERS_SIDX' (INDEX (DOMAIN))

Statistics
----------------------------------------------------------
          440  recursive calls
         3883  db block gets
         2539  consistent gets
            0  physical reads
            0  redo size
          657  bytes sent via SQL*Net to client
          508  bytes received via SQL*Net from client
            2  SQL*Net roundtrips to/from client
            3  sorts (memory)
            0  sorts (disk)
            5  rows processed
```

Spatial Operator with Multiple Hints in a SQL Statement with Two Tables:

```
SQL> SELECT /*+ ORDERED */ ct.id, ct.name
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_WITHIN_DISTANCE
  5  (ct.location, comp.location, 'DISTANCE=0.25 UNIT=MILE ' )='TRUE'
  6  ORDER BY ct.id ;

        ID                 NAME
---------- ------------------------------------
        25 BLAKE CONSTRUCTION
        28 COLONIAL PARKING
        34 HEWLETT-PACKARD DC GOV AFFAIRS
        41 MCGREGOR PRINTING
        48 POTOMAC ELECTRIC POWER
        50 SMITH HINCHMAN AND GRYLLS
       270 METRO-FARRAGUT NORTH STATION
       271 METRO-FARRAGUT WEST STATION
       468 SAFEWAY
       809 LINCOLN SUITES
       810 HOTEL LOMBARDY
      1044 MUSEUM OF THE THIRD DIMENSION
      1526 INTERNATIONAL FINANCE
      1538 MCKENNA AND CUNEO
      2195 STEVENS ELEMENTARY SCHOOL
      6326 HOTEL LOMBARDY
```

```
      7754 EXECUTIVE INN
      7762 PHILLIPS 66
      7789 SEVEN BUILDINGS
      7821 RENAISSANCE MAYFLOWER HOTEL
      8138 ST GREGORY HOTEL
      8382 EXXON
      8792 DESTINATION HOTEL & RESORTS

23 rows selected.


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=5 Card=32 Bytes=92
          48)

   1    0   SORT (ORDER BY) (Cost=5 Card=32 Bytes=9248)
   2    1     NESTED LOOPS (Cost=4 Card=32 Bytes=9248)
   3    2       TABLE ACCESS (BY INDEX ROWID) OF 'COMPETITORS' (TABLE)
           (Cost=2 Card=1 Bytes=137)

   4    3         INDEX (UNIQUE SCAN) OF 'COMPETITORS_PK' (INDEX (UNIQ
          UE)) (Cost=1 Card=1)

   5    2       TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS' (TABLE) (
          Cost=4 Card=32 Bytes=4864)

   6    5         DOMAIN INDEX OF 'CUSTOMERS_SIDX' (INDEX (DOMAIN))

Statistics
----------------------------------------------------------
        213  recursive calls
          4  db block gets
       1733  consistent gets
          0  physical reads
          0  redo size
       1253  bytes sent via SQL*Net to client
        519  bytes received via SQL*Net from client
          3  SQL*Net roundtrips to/from client
          3  sorts (memory)
          0  sorts (disk)
         23  rows processed
```

8.4 Advanced Spatial Index Features

This part covers some advanced spatial indexing features that are useful for large spatial repositories, which includes function-based indexing, online index rebuilds, three-dimensional indexing, parallel indexing, and local partitioned indexing.

- **Function-Based Spatial Indexes**
  Oracle allows create spatial indexes on any deterministic function that returns an SDO_GEOMETRY using existing columns of a table.

  The example is a creation of a deterministic function around the SDO_GCDR.GEOCODE_AS_GEOMETRY, which can use gcdr_geometry as the function in the function based spatial index.
  Creating a Deterministic Function to Return an SDO_GEOMETRY Using Address Attributes of the customers Table:

```
SQL> CREATE or REPLACE FUNCTION gcdr_geometry(
  2     street_number     varchar2,
  3     street_name       varchar2,
  4     city              varchar2,
  5     state             varchar2,
  6     postal_code       varchar2
  7  )
  8  RETURN MDSYS.SDO_GEOMETRY DETERMINISTIC is
  9  BEGIN
 10     RETURN (
 11        sdo_gcdr.geocode_as_geometry(
 12           'TEST',
 13           sdo_keywordarray(
 14              street_number || ' ' ||street_name ,
 15              city || ' ' || state || ' ' || postal_code
 16           ),
 17           'US')
 18     );
 19  END;
 20  /

Function created.
```

  Below example is a SQL statement with two invocations of the gcdr_geometry function, and the optimizer will evaluate this function just once if it is declared as DETERMINISTIC.

SQL Example Where a Deterministic Function is Evaluated Only Once Even When Called Twice:

```
SQL> SELECT
  2     gcdr_geometry(street_number,street_name,city,state,postal_code).sdo_point.x,
  3     gcdr_geometry(street_number,street_name,city,state,postal_code).sdo_point.y
  4  FROM customers
  5  WHERE id=1;

GCDR_GEOMETRY(STREET_NUMBER,STREET_NAME,CITY,STATE,POSTAL_CODE).SDO_POINT.X
--------------------------------------------------------------------------------
GCDR_GEOMETRY(STREET_NUMBER,STREET_NAME,CITY,STATE,POSTAL_CODE).SDO_POINT.Y
--------------------------------------------------------------------------------
                                                                      -76.97739
                                                                     38.8886508

Execution Plan
----------------------------------------------------------
   0        SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2 Card=1 Bytes=51)
   1    0     TABLE ACCESS (BY INDEX ROWID) OF 'CUSTOMERS' (TABLE) (Cost
           =2 Card=1 Bytes=51)
   2    1       INDEX (UNIQUE SCAN) OF 'CUSTOMERS_PK' (INDEX (UNIQUE)) (
           Cost=1 Card=1)

Statistics
----------------------------------------------------------
         594  recursive calls
           0  db block gets
        1726  consistent gets
           2  physical reads
           0  redo size
         630  bytes sent via SQL*Net to client
         508  bytes received via SQL*Net from client
           2  SQL*Net roundtrips to/from client
          55  sorts (memory)
           0  sorts (disk)
           1  rows processed
```

When populate the USER_SDO_GEOM_METADATA view that has the TABLE_NAME, COLUMN_NAME, DIMINFO, and SRID columns, the column_name need to be set as a pseudocolumn, which obtained using the GCDR_GEOMETRY function.

Below is an example of inserting information into the metadata.

Inserting the Metadata for a <table, function-based pseudo-column>:

```
SQL> INSERT INTO user_sdo_geom_metadata VALUES
  2  (
  3     'CUSTOMERS',
  4     'SPATIAL.GCDR_GEOMETRY(street_number,street_name,city,state,postal_code)',
  5     MDSYS.SDO_DIM_ARRAY
  6     (
  7        MDSYS.SDO_DIM_ELEMENT('X', -180, 180, 0.5),
  8        MDSYS.SDO_DIM_ELEMENT('Y', -90, 90, 0.5)
  9     ),
 10     8307
 11  );

1 row created.

Execution Plan
----------------------------------------------------------
   0         INSERT STATEMENT Optimizer=ALL_ROWS (Cost=1 Card=1)
   1      0     LOAD TABLE CONVENTIONAL OF 'USER_SDO_GEOM_METADATA'

Statistics
----------------------------------------------------------
            19   recursive calls
             8   db block gets
            16   consistent gets
             0   physical reads
          1272   redo size
           804   bytes sent via SQL*Net to client
          1053   bytes received via SQL*Net from client
             4   SQL*Net roundtrips to/from client
             1   sorts (memory)
             0   sorts (disk)
             1   rows processed
```

Creating a Spatial Index on a Function Returning an SDO_GEOMETRY:

```
SQL> CREATE INDEX customers_spatial_fun_idx ON customers
  2  (
  3     gcdr_geometry(street_number, street_name, city, state, postal_code)
  4  )
  5  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  6  PARAMETERS ('LAYER_GTYPE=POINT');

CREATE INDEX customers_spatial_fun_idx ON customers
*
```

```
ERROR at line 1:
ORA-29855: error occurred in the execution of ODCIINDEXCREATE routine
ORA-13203: failed to read USER_SDO_GEOM_METADATA view
ORA-13203: failed to read USER_SDO_GEOM_METADATA view
ORA-06512: at "MDSYS.SDO_INDEX_METHOD_10I", line 10
```

In this example:

- there is a error when creating a Spatial Index on a Function Returning an SDO_GEOMETRY, so the index can not be created

Therefore, the example of using the SDO_NN operator can not execute because it need to use a function-based index.

SDO_NN Operator Retrieving the Five Customers Nearest to Each Competitor Using the Function-Based Index:

```
SELECT /*+ ORDERED */ ct.id, ct.name, ct.customer_grade
FROM competitors comp, customers ct
WHERE comp.id=1
AND SDO_NN
(
  gcdr_geometry (ct.street_number,ct.street_name, ct.city, ct.state, ct.postal_code),
  comp.location,
  'SDO_NUM_RES=5'
)='TRUE'
ORDER BY ct.id;
```

- **Local Partitioned Spatial Indexes**

Table partitioning is an important feature of Oracle Spatial, which unable achieve scalability and manageability in large databases.

Creating Local Indexes on Partitioned Tables

In this case, a local index can be created for each partition by specifying the keyword LOCAL at the end of the CREATE INDEX statement and some optional partition-specific parameters. Indeed, there are many advantages of creating local indexes, such as:

- Manageability: it allows to rebuild the local index associated with a specific partition without affecting other partitions
- Scalability: using queries to specific partitions, improving performance

Below is the general SQL syntax for creating a local partitioned index.

Creating a Local Partitioned Spatial Index:

```
CREATE INDEX customers_sidx ON customers(location)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
[PARAMETERS ('parameter_string')]
LOCAL [PARAMETERS(sequence of 'partition-specific parameters')]
```

- **Parallel Indexing**

  The spatial indexes can be created in parallel, which the PARALLEL clause with an optional parallel_degree parameter needs to be specified at the end of the CREATE INDEX statement.

  Below is the general syntax of creating a parallel index.

  Creating a Spatial Index with the PARALLEL Keyword:

```
CREATE INDEX customers_sidx ON customers(location)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
[PARAMETERS ('parameter_string')] [LOCAL [Partition-specific parameters]]
PARALLEL [parallel_degree];
```

- **Spatial Joins**

  SDO_JOIN Table Function with SDO_RELATE Operator

  Below is an example of using RELATE operator only to retrieve all customers inside each competitor region:

  SDO_RELATE Operator Retrieving All Customers Inside (and Touching the Border of) Each Competitor Region:

```
SQL> SELECT COUNT(DISTINCT ct.id)
  2  FROM competitors comp, customers ct
  3    WHERE SDO_WITHIN_DISTANCE (ct.location, comp.location, 'DISTANCE=200
UNIT=METER ' )='TRUE';

COUNT(DISTINCTCT.ID)
--------------------
                1145
```

  In this example:

  - the query executes in a nested loop, performing the SDO_WITHIN_DISTANCE operation for each row in the competitors table

  The example rewrites the statement using the SDO_JOIN table function, where get the same account result.

  SDO_FILTER Operator Retrieving All Customers Whose MBRs Intersect Those of Competitor Regions:

```
SQL> SELECT COUNT(DISTINCT ct.id)
  2  FROM competitors comp, customers ct,
  3  TABLE
  4  (
  5    SDO_JOIN
  6    (
  7      'competitors', 'location',   -- first table and the SDO_GEOMETRY column
  8      'customers', 'location',     -- second table and the SDO_GEOMETRY column
  9      'DISTANCE=200 UNIT=METER'    -- specify mask relationship
 10    )
 11  )
```

```
 12   WHERE ct.rowid=jn.rowid2
 13   AND comp.rowid = jn.rowid1;

COUNT(DISTINCTCT.ID)
--------------------
                1145
```

SDO_JOIN Table Function with SDO_ FILTER Operator

Below is an example of using SDO_FILTER operator only to retrieve result.

SDO_FILTER Operator Retrieving All Customers Who's MBRs Intersect Those of Competitor Regions:

```
SQL> SELECT COUNT(DISTINCT ct.id)
  2  FROM competitors_sales_regions comp, customers ct
  3  WHERE SDO_FILTER(ct.location, comp.geom)='TRUE';

COUNT(DISTINCTCT.ID)
----------------------------------------
                2171
```

This example rewrite with SDO_JOIN function that uses two spatial indexes instead of one in SDO_FILTER to perform faster operation.

SDO_JOIN Operator Retrieving All Customers Who's MBRs Intersect the MBRs of Competitor:

```
SQL> SELECT COUNT(DISTINCT ct.id)
  2  FROM competitors_sales_regions comp, customers ct,
  3  TABLE
  4  (
  5     SDO_JOIN
  6     (
  7        'competitors_sales_regions', 'geom', -- first table and column
  8        'customers', 'location' -- second table and column
  9     )
 10  ) jn
 11  WHERE ct.rowid=jn.rowid2
 12  AND comp.rowid = jn.rowid1;

COUNT(DISTINCTCT.ID)
---------------------------------------
                2171
```

- **Three-Dimensional Analysis**

  Below is an example of inserting extent information in the USER_SDO_GEOM_METADATA view to create a three-dimensional spatial index on this column.

Inserting Metadata for the city_buildings Table (Extents in all Three Dimensions Are Entered):

```
SQL> insert into user_sdo_geom_metadata values (
  2  'CITY_BUILDINGS',
  3  'GEOM',
  4  SDO_DIM_ARRAY(
  5  SDO_DIM_ELEMENT('X', 29214140, 29219040, 0.05),
  6  SDO_DIM_ELEMENT('Y', 43364000, 43372640, 0.05),
  7  SDO_DIM_ELEMENT('Z', 0, 2000, 0.05)
  8  ),
  9  7407);

1 row created.
```

Below is the statement of creating the three-dimensional spatial index.

Creating an R-tree Index on Three-Dimensional City Data:

```
SQL> CREATE INDEX city_bldg_sidx ON city_buildings(geom)
  2  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  3  PARAMETERS ('SDO_INDX_DIMS=3');
```

In this example:

- the index is not able to be created, and a error message showed during the execution

Relationship analysis

The other usage of the SDO_FILTER operator is to identify the approximate set of buildings that intersect the helicopter trajectory, which is showed as the example.

Identifying the Buildings Intersecting a Helicopter Trajectory Using SDO_FILTER:

```
SQL> SELECT id FROM trip_route t, city_buildings c
  2  WHERE SDO_FILTER(c.geom, t.trajectory)='TRUE'
  3  ORDER BY id;

        ID
----------
         6
         8
        11
        12
        14

5 rows selected.
```

Below is an example of using the SDO_ANYINTERACT operator to get the exact set of buildings that intersect the query window using the three-dimensional geometries.

Identifying the Exact Set of Buildings Intersecting a Helicopter Trajectory Using

SDO_ANYINTERACT:

```
SQL> SELECT id FROM trip_route t, city_buildings c
  2   WHERE SDO_ANYINTERACT(c.geom, t.trajectory)='TRUE';

        ID
---------- -------
        14

1 row selected.
```

In this example:

- the return value shows the helicopter lands only on Building 14

Distance-Based Analysis

This example shows the SQL using the SDO_NUM_RES parameter to identify the three buildings that are close to the helicopter trajectory, and the distance to the trajectory is reported using the SDO_NN_DISTANCE ancillary operator.

Identifying the Five Closest Buildings to a Helicopter Trajectory:

```
SQL> SET numformat 99999
SQL> SELECT id, SDO_NN_DISTANCE(1) dist FROM trip_route t, city_buildings c
  2   WHERE SDO_NN(c.geom, t.trajectory, 'sdo_num_res=3', 1)='TRUE'
  3   ORDER BY dist;

    ID   DIST
------ ---------------
    14       0
    16     150
    10     357

3 rows selected.
```

This example is a compound system combines a horizontal coordinate system identified by CMPD_HORIZ_SRID and a vertical coordinate system identified by CMPD_VERT_SRID.

Identifying the Component SRIDs for Compound Coordinate System 7407:

```
SQL> SELECT srid, cmpd_horiz_srid, cmpd_vert_srid
  2   FROM sdo_coord_ref_sys
  3   WHERE srid=7407;

  SRID    CMPD_HORIZ_SRID    CMPD_VERT_SRID
------ --------------- ------------------------------------------------------------
  7407              32037              5702
```

In this example:

- the horizontal coordinate system (for first two dimensions): SRID is 32037
- vertical coordinate system (for the third dimension) : SRID is 5702

Below example examines the wktext for the horizontal SRID in the cs_srs table and search for the UNIT string.

Identifying the Units for Horizontal Coordinate System SRID 32037:

```
SQL> SELECT wktext FROM cs_srs WHERE srid=32037;

WKTEXT
--------------------------------------------------------------------------------
PROJCS["NAD27 / Texas North", GEOGCS [ "NAD27", DATUM ["North American Datum 1927
(EPSG ID 6267)", SPHEROID ["Clarke 1866 (EPSG ID 7008)", 6378206.4,
294.97869821390582076161053712319517 5418], -3, 142, 183, 0, 0, 0, 0], PRIMEM
[ "Greenwich", 0.000000 ], UNIT ["Decimal Degree", 0.01745329251994328]], PROJECTION
["Lambert Conformal Conic"], PARAMETER ["Latitude_Of_Origin", 34], PARAMETER
["Central_Meridian", -101.5], PARAMETER ["Standard_Parallel_1", 34.65], PARAMETER
["Standard_Parallel_2", 36.183333333333333333333333333333333333333], PARAMETER
["False_Easting", 2000000], PARAMETER ["False_Northing", 0], UNIT ["U.S.
Foot", .304800609601219202438404876809753619 5072]]
```

In this example:

- the horizontal SRID 32037: the UNIT is U.S. Foot

This example identifies the units for the vertical coordinate system by looking up the value after UoM.

Identifying the Units for Vertical Coordinate System SRID 5702:

```
SQL> SELECT coord_sys_name
  2  FROM sdo_coord_sys a, sdo_crs_vertical b
  3  WHERE b.srid=5702
  4  AND a.coord_sys_id = b.coord_sys_id;

COORD_SYS_NAME
--------------------------------------------------------------------------------
Gravity-related CS. Axis: height (H). Orientation: up.   UoM: ftUS.
```

This example shows the SQL for identifying buildings that are within 200 units of distance from the trajectory with the SDO_WITHIN_DISTANCE operator.

Identifying the Closest Buildings to a Helicopter Trajectory within 200 Feet:

```
SQL> SELECT id FROM trip_route t, city_buildings c
  2  WHERE SDO_WITHIN_DISTANCE(
  3     c.geom, t.trajectory,
  4   'distance = 200 unit=FOOT ') = 'TRUE'
  5  ORDER BY id;

     ID
------
     14
```

```
16


2 rows selected.
```

In this example:

- the closest buildings to a Helicopter Trajectory within 200 Feet are Building 14 and 16

9.0 GEOMETRY PROCESSING FUNCTIONS

Generally, the examples in this section provide experiments of using the geometry processing functions to solve real application problems by performing proximity analysis. In addition, compare with the spatial operators, these geometry functions do not need a spatial index, and provide more detailed analyses than the spatial operator. Moreover, the functions can be used individually or in combination to perform analysis on relationship, unions, intersections, extents, convex hulls of individual geometric objects and groups of geometric objects. Indeed, these functions assist in the business demographics analysis to help company with strategic decisions making, such as starting new businesses at appropriate locations.

9.1 Buffering Functions

Buffering function constructs a buffer around a specified geometric object or a set of geometric objects. For example, it can be used to create a quarter-mile buffer around a delivery site location, and the buffer geometry will be a circle around the delivery site with a radius of a quarter mile.

Below are two examples of constructing quarter-mile buffers around each branch location in the branches table, and the buffers are stored in the table called SALES_REGIONS and COMPETITORS_SALES_REGIONS.

Creating Buffers Around Branches:

```
-- create Sales_regions table
SQL> CREATE TABLE sales_regions AS
  2  SELECT id,
  3  SDO_GEOM.SDO_BUFFER(b.location, 0.25, 0.5, 'arc_tolerance=0.005 unit=mile') geom
  4  FROM branches b;

Table created.

-- Metadata for Sales_regions table
SQL> INSERT INTO user_sdo_geom_metadata
  2  SELECT 'SALES_REGIONS',
  3  'GEOM', diminfo, srid FROM user_sdo_geom_metadata
  4  WHERE table_name='BRANCHES';

  1 row created.

-- Index-creation for Sales_regions table
SQL> CREATE INDEX sr_sidx ON sales_regions(geom)
  2  INDEXTYPE IS mdsys.spatial_index;
```

```
Index created.
```

Creating Buffers Around Competitor Locations:

```
-- create COMPETITORS_SALES_REGIONS table
SQL> CREATE TABLE COMPETITORS_SALES_REGIONS AS
  2  SELECT id,
  3   SDO_GEOM.SDO_BUFFER(cmp.location, 0.25, 0.5, 'unit=mile arc_tolerance=0.005')
geom
  4  FROM competitors cmp;

Table created.

-- Metadata for Competitors_sales_regions table
SQL> INSERT INTO user_sdo_geom_metadata
  2  SELECT 'COMPETITORS_SALES_REGIONS',
  3  'GEOM', diminfo, srid FROM user_sdo_geom_metadata
  4  WHERE table_name='COMPETITORS';

1 row created.

-- Index-creation for Competitors_sales_regions table
SQL> CREATE INDEX cr_sidx ON competitors_sales_regions(geom)
  2  INDEXTYPE IS mdsys.spatial_index;

  Index created.
```

In these two examples:

- the first parameter (e.g. b.location): the geometry to be buffered
- the second parameter (e.g.0.25): specifies the buffer distance as 0.25
- the third parameter (e.g.0.5): specifies the tolerance unit for geodetic geometries is 0.5 meters
- the parameter_string (e.g. 'unit=mile arc_tolerance=0.005'): specifies the units for the buffer distance is mile, and an arc tolerance of 0.005 miles

9.2 Relationship Analysis Functions

Two types of analysis functions are covered in this part to demestrate the relationship between two SDO_GEOMETRY, which are:

1) SDO_DISTANCE function: determines the minimum distance between two geometries in the units specified
2) RELATE function: determines whether two geometries interact in any specified manner

- **SDO_DISTANCE**

  Below is an example of using SDO_DISTANCE function to identify the customers

within a quarter-mile radius of a competitor location.

Identifying Customers within a Quarter-Mile of a Competitor Location:

```
SQL> SELECT ct.id, ct.name
  2  FROM competitors comp, customers ct
  3  WHERE comp.id=1
  4  AND SDO_GEOM.SDO_DISTANCE(ct.location, comp.location, 0.5, 'unit=mile') < 0.25
  5  ORDER BY ct.id;

        ID NAME
---------- ------------------------------------
        25 BLAKE CONSTRUCTION
        28 COLONIAL PARKING
        34 HEWLETT-PACKARD DC GOV AFFAIRS
        41 MCGREGOR PRINTING
        48 POTOMAC ELECTRIC POWER
        50 SMITH HINCHMAN AND GRYLLS
       270 METRO-FARRAGUT NORTH STATION
       271 METRO-FARRAGUT WEST STATION
       468 SAFEWAY
       809 LINCOLN SUITES
       810 HOTEL LOMBARDY
      1044 MUSEUM OF THE THIRD DIMENSION
      1526 INTERNATIONAL FINANCE
      1538 MCKENNA AND CUNEO
      2195 STEVENS ELEMENTARY SCHOOL
      6326 HOTEL LOMBARDY
      7754 EXECUTIVE INN
      7762 PHILLIPS 66
      7789 SEVEN BUILDINGS
      7821 RENAISSANCE MAYFLOWER HOTEL
      8138 ST GREGORY HOTEL
      8382 EXXON
      8792 DESTINATION HOTEL & RESORTS

23 rows selected.
```

In this example:

- there are 23 customers within 0.25-mile radius of the competitor location

SDO_DISTANCE function with SDO_WITHIN_DISTANCE Operator

Below is an example of using SDO_DISTANCE function with SDO_WITHIN_DISTANCE operator to identify the same situation to the previous example. In this case, the search result is same, which there are 23 customers within a quarter-mile radius of a competitor location.

Using the SDO_DISTANCE Function with the SDO_WITHIN_DISTANCE Spatial Operator in SQL:

```
SQL> SELECT ct.id, ct.name,
  2      SDO_GEOM.SDO_DISTANCE(ct.location, comp.location, 0.5, 'unit=yard') distance
  3   FROM competitors comp, customers ct
  4   WHERE comp.id=1
  5         AND SDO_WITHIN_DISTANCE (ct.location, comp.location, 'distance=0.25
unit=mile')='TRUE'
  6   ORDER BY ct.id;

     ID NAME                                          DISTANCE
------ ------------------------------------ --------
     25 BLAKE CONSTRUCTION                             319
     28 COLONIAL PARKING                               398
     34 HEWLETT-PACKARD DC GOV AFFAIRS                 428
     41 MCGREGOR PRINTING                              350
     48 POTOMAC ELECTRIC POWER                         355
     50 SMITH HINCHMAN AND GRYLLS                      252
    270 METRO-FARRAGUT NORTH STATION                   345
    271 METRO-FARRAGUT WEST STATION                    272
    468 SAFEWAY                                        252
    809 LINCOLN SUITES                                 104
    810 HOTEL LOMBARDY                                 313
   1044 MUSEUM OF THE THIRD DIMENSION                  153
   1526 INTERNATIONAL FINANCE                          236
   1538 MCKENNA AND CUNEO                               97
   2195 STEVENS ELEMENTARY SCHOOL                      305
   6326 HOTEL LOMBARDY                                 329
   7754 EXECUTIVE INN                                  375
   7762 PHILLIPS 66                                    303
   7789 SEVEN BUILDINGS                                355
   7821 RENAISSANCE MAYFLOWER HOTEL                    322
   8138 ST GREGORY HOTEL                               359
   8382 EXXON                                          326
   8792 DESTINATION HOTEL & RESORTS                    159

23 rows selected.
```

Three-dimensional Geometries

Below is an example of using SDO_DISTANCE function on a three-dimensional object to determine the distance between two geometry locations.

Using the SDO_DISTANCE Function to Determine the Distance between Building 16 and the Helicopter Trajectory:

```
SQL> SELECT cbldg.id,
  2  SDO_GEOM.SDO_DISTANCE(cbldg.geom, tr.trajectory, 0.05, 'UNIT=foot') dist
  3  FROM trip_route tr, city_buildings cbldg
  4  WHERE cbldg.id=16;

     ID   DIST
------ -----
     16    150
```

The result shows the building 16 is 150.0003 feet away from the helicopter trajectory.

SDO_CLOSEST_POINTS

Below is an example of creating the SDO_CLOSEST_POINTS procedure, which can be used to determine the specific points on the building and the trajectory that are the closest

Obtaining the Closest Points on Building 16 and Helicopter Trajectory

set serverout on:

```
SQL> set serverout on
SQL> declare
  2     traj sdo_geometry;
  3     bldg16 sdo_geometry;
  4     dist number;
  5     trajpt sdo_geometry;
  6     bldg16pt sdo_geometry;
  7  begin
  8     select geom INTO bldg16 from city_buildings where id=16;
  9     select trajectory into traj from trip_route where rownum<=1;
 10     bldg16.sdo_srid:=null;          -- Workaround for Bug 6201938
 11     traj.sdo_srid:=null;            -- Workaround for Bug 6201938
 12     sdo_geom.sdo_closest_points(
 13        traj, bldg16, 0.05, 'UNIT=FOOT',
 14        dist, trajpt, bldg16pt);
 15     dbms_output.put_line('Distance= ' || TO_CHAR(dist));
 16     dbms_output.put_line('Pt on Trajectory:' ||
 17        TO_CHAR(trajpt.sdo_point.x) || ', ' ||
 18        TO_CHAR(trajpt.sdo_point.y) || ', ' ||
 19        TO_CHAR(trajpt.sdo_point.z));
 20     dbms_output.put_line('Pt on Bldg16:' ||
 21        TO_CHAR(bldg16pt.sdo_point.x) || ', ' ||
 22        TO_CHAR(bldg16pt.sdo_point.y) || ', ' ||
 23        TO_CHAR(bldg16pt.sdo_point.z));
 24  end;
 25  /
```

```
Distance= 150
Pt on Trajectory:29729872.8, 43920828.9, 650
Pt on Bldg16:29729872.8, 43920828.9, 500


PL/SQL procedure successfully completed.
```

In this example:

- the procedure returns a distance of 150 feet and the closest points on building 16 and the helicopter trajectory that have this distance

- **RELATE**

The following examples show how to use the SDO_RELATE operator and the RELATE function in the SDO_GEOM package to perform relationship analysis to identify customers inside these sales regions and competitor regions.

Below is an example of using the RELATE function to perform proximity analyses using the specified buffer zones around branches and competitors, which can identify all customers who are in competitors_sales_regions.

Identifying Customers in a Competitor Sales Region:

```
SQL> SELECT ct.id, ct.name
  2   FROM customers ct, competitors_sales_regions comp
  3   WHERE SDO_GEOM.RELATE (ct.location, 'INSIDE', comp.geom, 0.5) = 'INSIDE'
  4   AND comp.id=1
  5   ORDER BY ct.id;

     ID NAME
------ ------------------------------------
     25 BLAKE CONSTRUCTION
     28 COLONIAL PARKING
     34 HEWLETT-PACKARD DC GOV AFFAIRS
     41 MCGREGOR PRINTING
     48 POTOMAC ELECTRIC POWER
     50 SMITH HINCHMAN AND GRYLLS
    270 METRO-FARRAGUT NORTH STATION
    271 METRO-FARRAGUT WEST STATION
    468 SAFEWAY
    809 LINCOLN SUITES
    810 HOTEL LOMBARDY
   1044 MUSEUM OF THE THIRD DIMENSION
   1526 INTERNATIONAL FINANCE
   1538 MCKENNA AND CUNEO
   2195 STEVENS ELEMENTARY SCHOOL
   6326 HOTEL LOMBARDY
   7754 EXECUTIVE INN
   7762 PHILLIPS 66
```

```
  7789 SEVEN BUILDINGS
  7821 RENAISSANCE MAYFLOWER HOTEL
  8138 ST GREGORY HOTEL
  8382 EXXON
  8792     DESTINATION HOTEL & RESORTS

23 rows selected.
```

In this example:

- the result shows there are 23 customers at a distance of less than 0.25 miles from the specified competitor (id=1)

The follow example specify the ANYINTERACT mask within the RELATE function to perform the previous situation.

Identifying Customers Who Interact with a Competitor's Sales Region:

```
SQL> SELECT ct.id, ct.name
  2   FROM customers ct, competitors_sales_regions comp
  3    WHERE SDO_GEOM.RELATE (ct.location, 'ANYINTERACT', comp.geom, 0.5) =
'TRUE'
  4   AND comp.id=1
  5   ORDER BY ct.id;

     ID NAME
------ ----------------------------------
     25 BLAKE CONSTRUCTION
     28 COLONIAL PARKING
     34 HEWLETT-PACKARD DC GOV AFFAIRS
     41 MCGREGOR PRINTING
     48 POTOMAC ELECTRIC POWER
     50 SMITH HINCHMAN AND GRYLLS
    270 METRO-FARRAGUT NORTH STATION
    271 METRO-FARRAGUT WEST STATION
    468 SAFEWAY
    809 LINCOLN SUITES
    810 HOTEL LOMBARDY
   1044 MUSEUM OF THE THIRD DIMENSION
   1526 INTERNATIONAL FINANCE
   1538 MCKENNA AND CUNEO
   2195 STEVENS ELEMENTARY SCHOOL
   6326 HOTEL LOMBARDY
   7754 EXECUTIVE INN
   7762 PHILLIPS 66
   7789 SEVEN BUILDINGS
   7821 RENAISSANCE MAYFLOWER HOTEL
   8138 ST GREGORY HOTEL
```

```
  8382 EXXON
  8792 DESTINATION HOTEL & RESORTS


23 rows selected.
```

In this example:

- the function is compared with 'TRUE' instead of 'ANYINTERACT'
- the RELATE function returns 'TRUE' if the geometries intersect and the ANYINTERACT mask is specified

In addition, the SDO_GEOM.RELATE functions to operate on subsets of geometries. Below is an example of using the SDO_GEOM.RELATE function to determine the relationship for each row that is returned.

Identifying Customers in Quarter-Mile Buffer around a Competitor:

```
SQL> SELECT ct.id, ct.name,
  2  SDO_GEOM.RELATE (ct.location, 'DETERMINE', comp.geom, 0.5) relationship
  3  FROM customers ct, competitors_sales_regions comp
  4  WHERE comp.id=1
  5  AND SDO_RELATE(ct.location, comp.geom, 'mask=anyinteract')='TRUE';

    ID    NAME                                RELATIONSHIP
------ ---------------------------------------------------------------------------------------------------------------
    25   BLAKE CONSTRUCTION                  INSIDE
  7821   RENAISSANCE MAYFLOWER HOTEL         INSIDE
  8138   ST GREGORY HOTEL                    INSIDE
    28   COLONIAL PARKING                    INSIDE
    41   MCGREGOR PRINTING                   INSIDE
  2195   STEVENS ELEMENTARY SCHOOL           INSIDE
   468   SAFEWAY                             INSIDE
   810   HOTEL LOMBARDY                      INSIDE
  8792   DESTINATION HOTEL & RESORTS         INSIDE
  1044   MUSEUM OF THE THIRD DIMENSION       INSIDE
  6326   HOTEL LOMBARDY                      INSIDE
  8382   EXXON                               INSIDE
  1538   MCKENNA AND CUNEO                   INSIDE
    48   POTOMAC ELECTRIC POWER              INSIDE
  7789   SEVEN BUILDINGS                     INSIDE
   809   LINCOLN SUITES                      INSIDE
  1526   INTERNATIONAL FINANCE               INSIDE
    50   SMITH HINCHMAN AND GRYLLS           INSIDE
   271   METRO-FARRAGUT WEST STATION         INSIDE
  7762   PHILLIPS 66                         INSIDE
   270   METRO-FARRAGUT NORTH STATION        INSIDE
  7754   EXECUTIVE INN                       INSIDE
    34   HEWLETT-PACKARD DC GOV AFFAIRS      INSIDE
```

```
23 rows selected.
```

RELATE function with SDO_WITHIN_ RELATE Operator

Below is an example of using RELATE function with SDO_ RELATE operator to retrieve all sales regions that intersect a sales region with id=51.

RELATE Function Complementing the SDO_RELATE Operator:

```
SQL> SELECT sra.id,
  2  SDO_GEOM.RELATE(sra.geom, 'DETERMINE', srb.geom, 0.5) relationship
  3  FROM sales_region srb, sales_region sra
  4  WHERE srb.id=51
  5  AND sra.id<>51
  6  AND SDO_RELATE (
  7     sra.geom, srb.geom,
  8    'mask=TOUCH+OVERLAPBDYDISJOINT+OVERLAPBDYINTERSECT'
  9  ) = 'TRUE'
 10  ORDER BY sra.id;

    ID                             RELATIONSHIP
------------------------------------------------------------------------------------
    43                         OVERLAPBDYINTERSECT
    50                         OVERLAPBDYINTERSECT
    54                         OVERLAPBDYINTERSECT
    63                         OVERLAPBDYINTERSECT
    66                         OVERLAPBDYINTERSECT
    69                         OVERLAPBDYINTERSECT
    72                         OVERLAPBDYINTERSECT
    75                         OVERLAPBDYINTERSECT
    76                         TOUCH

9 rows selected.
```

In this example:

- the result shows there are 9 the sales regions that overlap with a specific sales region (with id=51)

Three-dimensional Geometries

Below is an example using the RELATE function with specified ANYINTERACT mask to determine which buildings intersect the helicopter trajectory using the SDO_GEOM.RELATE function.

Identifying Buildings That Intersect the Helicopter Trajectory:

```
SQL> SELECT cbldg.id
  2  FROM city_buildings cbldg, trip_route tr
  3   WHERE SDO_GEOM.RELATE (cbldg.geom, 'ANYINTERACT', tr.trajectory, 0.5) =
```

```
'TRUE';

      ID
------
      14

1 row selected.
```

9.3 Geometry Combination Functions

- **SDO_INTERSECTION**

  Below is an example that identifies the intersection geometry for each pair of overlapping sales regions. The sales_intersection_zones table stores these intersection regions.

  SDO_INTERSECTION of Two Geometriest:

```
SQL> CREATE TABLE sales_intersection_zones AS
  2  SELECT sra.id id1, srb.id id2,
  3  SDO_GEOM.SDO_INTERSECTION(sra.geom, srb.geom, 0.5) intsxn_geom
  4  FROM sales_region srb, sales_region sra
  5  WHERE sra.id<> srb.id
  6  AND SDO_RELATE(sra.geom, srb.geom, 'mask=anyinteract' )='TRUE' ;

Table created.
```

  Below example performs a checking to shows the ids of sales regions that overlap with sales region 51.

  Sales Regions Intersecting the Sales Region with id=51:

```
SQL> SELECT id2 FROM sales_intersection_zones WHERE id1=51;

    ID2
------
      43
      50
      54
      63
      66
      69
      72
      75
      76

9 rows selected.
```

Below example shows using the geometry to identify customers in the intersection area of two specific sales regions with ids 51 and 43.

Identifying Customers in sales_ intersection_zones:

```
SQL> SELECT count(*)
  2  FROM sales_intersection_zones siz, customers ct
  3  WHERE siz.id1=51 AND siz.id2=43
  4  AND SDO_RELATE(ct.location, siz.intsxn_geom, 'mask=anyinteract')='TRUE';

COUNT(*)
--------
         2
```

In this example:

- the result indicates that there are only two customers common to both sales regions

Below example performs the preceding intersection analysis without materializing the intersection regions in a separate table, and it identifies customers in the intersection of sales regions 51 and 43.

Identifying Customers in the Intersection of Sales Regions 51 and 43lt:

```
SQL> SELECT COUNT(*)
  2  FROM customers ct
  3  WHERE SDO_RELATE
  4  (
  5     ct.location,
  6     (
  7        SELECT SDO_GEOM.SDO_INTERSECTION(sra.geom, srb.geom, 0.5)
  8        FROM sales_regions sra, sales_regions srb
  9        WHERE sra.id = 51 and srb.id = 43
 10     ),
 11     'mask=anyinteract'
 12  )='TRUE';

COUNT(*)
--------
         2
```

- **SDO_UNION**

The below examples show using SDO_UNION function to compute the geometry covered by two sales regions. Firstly, it can be used to result union geometry to identify the total number of customers

SDO_UNION of Two Geometries:

```
SQL> SELECT count(*)
  2  FROM
```

```
  3  (
  4     SELECT SDO_GEOM.SDO_UNION (sra.geom, srb.geom, 0.5) geom
  5     FROM sales_regions srb, sales_regions sra
  6     WHERE sra.id=51 and srb.id=43
  7  ) srb, customers sra
  8  WHERE SDO_RELATE(sra.location, srb.geom, 'mask=anyinteract')='TRUE';

COUNT(*)
--------
       124
```

In this example:

- the number of customers returned is 124
- base on the previous example, which show there are 122 customers who are not common to both sales regions

Below is an example of creating a procedure to perform a union of all the sale regions, in order to identify the coverage of all businesses in a certain area.

Coverage of the Sales Regions: Performing a Union of All the Sales Regions:

```
SQL> DECLARE
  2     coverage SDO_GEOMETRY := NULL;
  3  BEGIN
  4     FOR g IN (SELECT geom FROM sales_regions) LOOP
  5        coverage := SDO_GEOM.SDO_UNION(coverage, g.geom, 0.5);
  6     END LOOP;
  7     INSERT INTO sales_region_coverage values (coverage);
  8     COMMIT;
  9  END;
 10  /

PL/SQL procedure successfully completed.
```

- **SDO_DIFFERENCE**

The SDO_DIFFERENCE function subtracts the second geometry from the first geometry, which returns the region that is exclusive to the first geometry.

The follow example shows using the SDO_DIFFERENCE function to create a competitor region 2 which used to compare the sales regions with the competitor regions

SDO_DIFFERENCE of Competitor Region 2 with Sales Region 6:

```
SQL> CREATE TABLE exclusive_region_for_comp_2 AS
  2  SELECT SDO_GEOM.SDO_DIFFERENCE(sr.geom, csr.geom, 0.5) geom
  3  FROM sales_regions sr, competitors_sales_regions csr
  4  WHERE csr.id=2 and sr.id=6 ;

Table created.
```

Below example shows using this region to identify customers in this exclusive zone, which the customer is targeted on special promotions to wean them from the specific competitor (id=2).

Identifying Customers in an Exclusive Zone of a Competitor:

```
SQL> SELECT ct.id, ct.name
  2  FROM exclusive_region_for_comp_2 excl, customers ct
  3  WHERE SDO_RELATE(ct.location, excl.geom, 'mask=anyinteract')='TRUE'
  4  ORDER BY ct.id;

     ID NAME
------ ------------------------------------
    510 EXXON
    797 GEORGETOWN INN
   1379 KEY THEATRE
   2152 HYDE ELEMENTARY SCHOOL
   6954 ST JOHN'S CHURCH
   7177 THE SHOPS AT GEORGETOWN PARK
   7178 THE SHOPS AT GEORGETOWN PARK
   7385 FRANCIS SCOTT KEY MEMORIAL PARK
   7388 FRANCIS SCOTT KEY MEMORIAL PARK
   7393 GEORGETOWN UNIVERSITY
   7402 HALCYON HOUSE

     ID NAME
------ ------------------------------------
   7595 THE EXORCIST STEPS
   8812 GEORGETOWN EXXON

13 rows selected.
```

Below is an example using a combined function to obtain the same 20 customers as above situation.

Combining Listings 9-22 and 9-23:

```
QL> SELECT ct.id, ct.name
  2  FROM sales_regions sr, competitors_sales_regions csr, customers ct
  3  WHERE csr.id=2 AND sr.id=6
  4  AND SDO_RELATE
  5  (
  6     ct.location,
  7     SDO_GEOM.SDO_DIFFERENCE(csr.geom, sr.geom, 0.5),
  8     'mask=anyinteract'
  9  )='TRUE'
 10  ORDER BY ct.id;
```

```
        ID NAME
------ ------------------------------------
          510 EXXON
       797 GEORGETOWN INN
      1379 KEY THEATRE
      2152 HYDE ELEMENTARY SCHOOL
      6954 ST JOHN'S CHURCH
      7177 THE SHOPS AT GEORGETOWN PARK
      7178 THE SHOPS AT GEORGETOWN PARK
      7385 FRANCIS SCOTT KEY MEMORIAL PARK
      7388 FRANCIS SCOTT KEY MEMORIAL PARK
      7393 GEORGETOWN UNIVERSITY
      7402 HALCYON HOUSE

        ID NAME
------ ------------------------------------
      7595 THE EXORCIST STEPS
      8812 GEORGETOWN EXXON

13 rows selected.
```

- **SDO_XOR**
  The SDO_DIFFERENCE of the SDO_UNION and the SDO_INTERSECTION of the two geometries can be rewrite in SDO_XOR function, which can used to identify whether two overlapping sales regions need to be merged.
  Below shows an example of SDO_XOR function.
  SDO_XOR of Sales Regions 43 and 51 to Identify Customers Who Are Not Shared Between Them:

```
SQL> SELECT count(*)
  2  FROM
  3  (
  4     SELECT SDO_GEOM.SDO_XOR (sra.geom, srb.geom, 0.5) geom
  5     FROM sales_regions srb, sales_regions sra
  6     WHERE sra.id=51 and srb.id=43
  7  ) srb, customers sra
  8  WHERE SDO_RELATE(sra.location, srb.geom, 'mask=anyinteract')='TRUE';

COUNT(*)
--------
       122
```

  In this example:
  - the result shows the number in the SDO_XOR (122 customer) is close to that in the SDO_UNION (124 customer), which means these two sales regions have overlapping and few common customers

9.4 Geometric Analysis Functions

In addition, the Geometric Analysis Functions are used to perform further analysis on individual geometries, such as computing the area of the intersection region for each pair of overlapping sales regions.

- **Area, Length, and Volume Functions**
  The functions for calculating the area, length, or volume of an input SDO_GEOMETRY to Accuracy of Area and Length Computations for Geodetic Data These objects can also be used on either a two-dimensional geometry or a three-dimensional geometry.

  SDO_AREA
  Below is an example of using SDO_AREA function to computes the area of an SDO_GEOMETRY object, which the area of the intersection of sales region 51 with sales region 43 in the sales_regions table
  Area of the Intersection Region of Sales Region 43 and Sales Region 51:

```
SQL> SELECT SDO_GEOM.SDO_AREA (
```

```
  2      SDO_GEOM.SDO_INTERSECTION(sra.geom, srb.geom, 0.5), 0.5, ' unit=sq_yard ' )
area
  3   FROM sales_regions srb, sales_regions sra
  4   WHERE sra.id=51
  5   AND srb.id=43;

   AREA
------
 26243
```

Below is an example of inserting the building into the city_buildings table.

Inserting a New Building of Dimensions 200 Feet by 200 Feet by 400 Feet into city_buildings:

```
SQL> insert into city_buildings (id, geom)
  2   values (
  3      1, -- ID of the building
  4      sdo_geometry(3008, 7407, null,
  5         sdo_elem_info_array(1,1007,3), -- 3 represents a Solid Box representation using just
the co
rner points
  6         sdo_ordinate_array(
  7            27731202, 42239124, 0,       -- Min values for x, y, z
  8            27731402, 42239324, 400      -- Max values for x, y, z
  9         )
 10      )
 11   );

1 row created.

SQL> commit;

Commit complete.
```

The below example illustrates the computation of surface area for building 1 of the city_buildings table, which the area function sums the area of each of the six faces of the solid.

Surface Area of Building 1 (in Default Units of Square Feet):

```
SQL> SELECT id, SDO_GEOM.SDO_AREA(geom, 0.05) SURFACE_AREA
  2   FROM city_buildings
  3   WHERE id=1;

     ID SURFACE_AREA
------ ------------
      1       400000
```

SDO_LENGTH

This function is used to identify the length for a line string and the perimeter for a polygon, surface, or solid.

Below example shows the interstates that are not more than 1 mile in length.

Identifying Interstates Shorter Than 1 Mile:

```
SQL> SELECT interstate
  2  FROM us_interstates
  3  WHERE SDO_GEOM.SDO_LENGTH(geom, 0.5, 'unit=mile') < 1;

INTERSTATE
----------------------------------
I10/I45
I40/I65
I30/I35E
I55B
I564
I71/I670
I670/315
I96S
I90/I87
I94S
I94/I35E
I94/I35W

12 rows selected.
```

Below are two examples of using sdo_length function to determine the total length of the connecting edges that make up the geometry in a three-dimensional surfaces and solids.

Length of the Building 1 (in Units of Feet) with count_shared_edges Set to 1:

```
SQL> SELECT SDO_GEOM.SDO_LENGTH(
  2     geom,              -- input geometry
  3     0.05,              -- tolerance value
  4     'UNIT=FOOT',   -- units parameter
  5     1                  -- count_shared_edges only once
  6  ) LENGTH
  7  FROM city_buildings
  8  WHERE id=1;

LENGTH
------
  3200
```

Length of the Building 1 (in Units of Feet) with count_shared_edges Set to 2:

```
SQL> SELECT SDO_GEOM.SDO_LENGTH(
  2      geom,              -- input geometry
  3      0.05,              -- tolerance value
  4      'UNIT=FOOT',    -- units parameter
  5      2                  -- count_shared_edges only once
  6   ) LENGTH
  7   FROM city_buildings
  8   WHERE id=1;

LENGTH
------
  6400
```

SDO_VOLUME

The sdo_volume function is used in determining the volume of a three-dimensional solid or a multisolid geometry with an input geometry and a tolerance value.

Below example computes the volume (in default units of cubic feet) for building 1 in the city_buildings table

Volume of Building 1 (in Default Units of Cubic Feet):

```
SQL> set numwidth 15
SQL> SELECT SDO_GEOM.SDO_VOLUME(
  2      GEOM, -- INPUT GEOMETRY
  3      0.05 -- TOLERANCE VALUE
  4   ) VOLUME
  5   FROM city_buildings
  6   WHERE id=1;

VOLUME
------
16000000
```

- **MBR Functions**

MBR is short for the ‘minimum bounding rectangle’, which used to demonstrate the lower bound and upper bound of a geometry in each dimension. In this case, the MBR functions in Oracle Spatial are used to compute the MBR and associated components, and it can be used to perform simple geometric analyses such as computing the centroid or computing the convex hull.

SDO_MBR

The SDO_MBR function takes an SDO_GEOMETRY as an argument and computes the MBR for the geometry, which returns an SDO_GEOMETRY object.

Below is an example on two-demational objects that the function is used to get the MBR for a specific sales region in the sales_regions table.

Computing the MBR of a Geometry:

```
SQL> SELECT SDO_GEOM.SDO_MBR(sr.geom) mbr FROM sales_regions sr
  2  WHERE sr.id=1;

MBR(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3),
SDO_ORDINATE_ARRAY(-77, 39, -77, 39))
```

Below is an example of using the SDO_MBR function with three-dimensional puts, which returns the minimum and maximum values in all three dimensions as a three-dimensional geometry.

Example for Building 1 of the City_buildings Table to *Computing the Extent of a Three-Dimensional Object:*

```
SQL> SELECT SDO_GEOM.SDO_MBR(geom) extent
  2  FROM city_buildings cbldg
  3  WHERE id=1;

EXTENT(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(3008, 7407, NULL, SDO_ELEM_INFO_ARRAY(1, 1007, 3),
SDO_ORDINATE_ARRAY(27731202, 42239124, 0, 27731402, 42239324, 400))
```

SDO_MIN_ORDINATE and SDO_MAX_MBR_ORDINATE

The SDO_MIN_MBR_ORDINATE and SDO_MAX_MBR_ORDINATE functions can be used to return the minimum and maximum ordinates of a geometry in a specified dimension, respectively.

Below is an example shows getting the extent in the first dimension using these two functions on a two-demational objects.

Obtaining the MIN_ORDINATEs and MAX_ORDINATEs in a Specific Dimension:

```
SQL> SELECT SDO_GEOM.SDO_MIN_MBR_ORDINATE(sr.geom, 1) min_extent,
  2  SDO_GEOM.SDO_MAX_MBR_ORDINATE(sr.geom, 1) max_extent
  3  FROM sales_regions sr WHERE sr.id=1;

MIN_EXTENT MAX_EXTENT
---------- ----------
       -77        -77
```

Below is an example of using The SDO_MIN_MBR_ORDINATE and SDO_MAX_MBR_ORDINATE functions on three-dimensional objects for building 1 in the city_buildings table.

Obtaining MIN_ORDINATE and MAX_ORDINATE in the Third Dimension:

```
SQL> SELECT SDO_GEOM.SDO_MIN_MBR_ORDINATE(geom, 3) min_extent,
  2  SDO_GEOM.SDO_MAX_MBR_ORDINATE(geom, 3) max_extent
  3  FROM city_buildings cbldg
  4  WHERE id=1;

MIN_EXTENT MAX_EXTENT
---------- ----------
         0        400
```

SDO_CONVEXHULL

The convex hull of geometry is the smallest convex set that contains all of that geometry. Moreover, the SDO_CONVEXHULL function computes the convex hull of an SDO_GEOMETRY, which traditionally used to calculate the smallest convex geometry covering a set of points or a set of polygons.

The example in below illustrates the convex hull for the state of New Hampshire.

Computing the Convex Hull for the State of New Hampshire:

```
SQL> SELECT SDO_GEOM.SDO_CONVEXHULL(st.geom, 0.5) cvxhl
  2  FROM us_states st
  3  WHERE st.state_abrv='NH';

CVXHL(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARR
AY(-71, 43, -71, 43, -71, 43, -71, 43, -71, 43, -71, 43, -71, 45, -71, 45, -71, 45, -71, 45, -72, 44,
-72, 44, -72, 44, -72, 44, -73, 43, -73, 43, -73, 43, -73, 43, -73, 43, -72, 43, -72, 43, -72, 43, -72,
43, -72, 43, -72, 43, -72, 43, -72, 43, -71, 43, -71, 43, -71, 43))
```

SDO_CENTRIOD

The SDO_CENTROID function computes the geometric centroid of an SDO_GEOMETRY object.

The example in below shows the method of computing the centroid for the state of New Hampshire using the SDO_CENTROID function.

Computing the Centroid for the State of New Hampshire:

```
SQL> SELECT SDO_GEOM.SDO_CENTROID(st.geom, 0.5) ctrd
  2  FROM us_states st WHERE st.state_abrv='NH';

CTRD(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-72, 44, NULL), NULL, NULL)
```

SDO_POINTONSURFACE

Below is an example of using SDO_POINTONSURFACE function to obtaining a point on the surface of the geometry of the State of Massachusetts on two-dimensional geometries.

Obtaining a Point on the Surface of the Geometry:

```
SQL> SELECT SDO_GEOM.SDO_POINTONSURFACE(st.geom, 0.5) pt
  2  FROM us_states st
  3  WHERE state_abrv='MA';


PT(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-73, 43, NULL), NULL, NULL)
```

Below is an example of using SDO_POINTONSURFACE function to obtaining a point on the surface of the geometry of the State of Massachusetts on three-dimensional geometries.

Obtaining a Point on Building 1:

```
SQL> SELECT SDO_GEOM.SDO_POINTONSURFACE(geom, 0.05) pt
  2  FROM city_buildings cbldg
  3  WHERE id=1;


PT(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(3001, 7407, SDO_POINT_TYPE(27731202, 42239124, 0), NULL, NULL)
```

9.5 Aggregate Functions

- **Aggregate MBR Function**

  The SDO_AGGR_MBR function is used to compute a collection of the MBR.

  Below example computes the aggregate MBR for all the locations in the branches table illustrates using this function.

  Finding the Extent of a Set of Geometries Using SDO_AGGR_MBR:

```
SQL> SELECT SDO_AGGR_MBR(location) extent FROM branches;

EXTENT(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3),
SDO_ORDINATE_ARRAY(-122, 38, -77, 39))
```

- **SDO_AGGR_UNION**

  The SDO_AGGR_UNION function is used to compute the union of a set of geometries, which returns an SDO_GEOMETRY object.

  The below example is the creation of a union of all the locations in the branches table to identify the coverage of stores.

  Finding the Coverage of Branch Locations Using SDO_AGGR_UNION:

```
SQL> SELECT SDO_AGGR_UNION(SDOAGGRTYPE(location, 0.5)) coverage
  2  FROM branches;

COVERAGE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2005, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1, 77),
SDO_ORDINATE_ARRAY (-122, 38, -122, 38, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77,
39, -77,39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39,
-77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -122, 38,
-122, 38, -122, 38, -77, 39, -77, 39, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38))
```

  Below example shows how to select a union of three sales regions, which there are two overlapping polygons, sales regions 51 and 43, and a third disjoint, region 2.

  Union of Three Sales Regions (ids 43, 51, and 2):

```
SQL> SELECT SDO_AGGR_UNION(SDOAGGRTYPE(geom, 0.5)) union_geom
  2  FROM sales_regions
```

```
  3  WHERE id=51 or id=43 or id=2 ;

UNION_GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES
--------------------------------------------------------------------------------
SDO_GEOMETRY(2007, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1, 35, 1003, 1),
SDO_ORDINATE_ARRAY(-77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39,
-77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38))

1 row selected.
```

Below is an example shows how to compute the union of all the sales_regions.
Union of All Sales Regions to Obtain Business Coverage:

```
SQL> SELECT SDO_AGGR_UNION(SDOAGGRTYPE(geom, 0.5)) coverage
  2  FROM sales_regions;

COVERAGE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39,
-77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77,
39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39,
-77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77,
39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39,
-77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77,
39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39,
-77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77,
39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77

……

COVERAGE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38, -122, 38,
-122, 38, -122, 38, -122, 38, -122, 38))

11 row selected.
```

SDO_AGGR_CONVEHULL

The SDO_AGGR_CONVEXHULL function can compute the convex hull from the set of sales_regions.

This example shows the code for computing the coverage of sales regions using the SDO_AGGR_CONVEXHULL function.

Finding the Coverage of sales_regions Using SDO_AGGR_CONVEXHULL:

```
SQL> SELECT SDO_AGGR_CONVEXHULL(SDOAGGRTYPE(geom, 0.5)) coverage
  2  FROM sales_regions;

COVERAGE(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2003, 8307, NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARR
AY(-122, 38, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77, 39, -77,
 39, -77, 39, -77, 39, -77, 39, -77, 39, -123, 38, -123, 38, -123, 38, -123, 38,
 -123, 38, -123, 38, -123, 38, -123, 38, -123, 38, -123, 38, -123, 38, -123, 38,
 -122, 38, -122, 38))
```

SDO_AGGR_CENTROID

The SDO_AGGR_CENTROID function can used to compute the centroid for an arbitrary group of customers which is showing in below.

Finding the Centroid of Customer Locations Using SDO_AGGR_CENTROID:

```
SQL> SELECT SDO_AGGR_CENTROID(SDOAGGRTYPE(location, 0.5)) ctrd
  2  FROM customers
  3  WHERE id>100;

CTRD(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO,
SDO_ORDINATES)
--------------------------------------------------------------------------------
SDO_GEOMETRY(2001, 8307, SDO_POINT_TYPE(-103, 38, NULL), NULL, NULL)
```

10.0 NETWORK MODELING

In addition, a network is a useful method of modeling information, and its concept can be used in spatial data modeling. Spatial network in Oracle Spatial are defined as SDO_GEOMETRY objects, which represents points and lines. Moreover, various types of networks are used to solve many different problems, such as Road Network, Utility Networks and Project Networks. In this section, the creation and verification of Spatial Networks are experiment with examples.

10.1 Defining Networks

In this part, several ways to define the data structures for a network are examined using procedures and functions of the SDO_NET package.

- **“Automatic” Network Definition**

  In general, the “automatic” creation method is easy to use and it automatically populates the network metadata. However, it not flexible, and it give limited control over the actual structuring of the tables and no control over the physical storage (tablespaces, space management, partitioning, and so on). Moreover, the CREATE_SDO_NETWORK or CREATE_LOGICAL_NETWORK procedure can be used in to create the structures of the network.

  Below is an example of creating a spatial network called US_ROADS.

  Creating a Spatial Network Using Default Table Names:

```
SQL> BEGIN
  2     SDO_NET.CREATE_SDO_NETWORK (
  3        NETWORK                       => 'US_ROADS',
  4        NO_OF_HIERARCHY_LEVELS   => 1,
  5        IS_DIRECTED                  => TRUE,
  6        NODE_WITH_COST               => FALSE
  7     );
  8  END;
  9  /

PL/SQL procedure successfully completed.

SQL> EXEC SDO_NET.CREATE_SDO_NETWORK ('US_ROADS',1,TRUE,FALSE)

PL/SQL procedure successfully completed.
```

  In this example:

  - four network tables called US_ROADS_NODE$, US_ROADS_LINK$, US_ROADS_PATH$, and US_ROADS_PLINK$ are created by the procedure

- the procedure also adds information to the network metadata

The following example shows the structure of the resulting tables in the US_ROADS network.

Structure of Default Network Tables:

```
-- US_ROADS_NODE table
SQL> describe US_ROADS_NODE$
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 NODE_ID                                   NOT NULL NUMBER
 NODE_NAME                                          VARCHAR2(200)
 NODE_TYPE                                          VARCHAR2(200)
 ACTIVE                                             VARCHAR2(1)
 PARTITION_ID                                       NUMBER
 GEOMETRY                                           MDSYS.SDO_GEOMETRY

-- US_ROADS_LINK table
SQL> describe US_ROADS_LINK$
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 LINK_ID                                   NOT NULL NUMBER
 LINK_NAME                                          VARCHAR2(200)
 START_NODE_ID                             NOT NULL NUMBER
 END_NODE_ID                               NOT NULL NUMBER
 LINK_TYPE                                          VARCHAR2(200)
 ACTIVE                                             VARCHAR2(1)
 LINK_LEVEL                                         NUMBER
 GEOMETRY                                           MDSYS.SDO_GEOMETRY
 COST                                               NUMBER
 BIDIRECTED                                         VARCHAR2(1)

-- US_ROADS_PATH table
SQL> describe US_ROADS_PATH$
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PATH_ID                                   NOT NULL NUMBER
 PATH_NAME                                          VARCHAR2(200)
 PATH_TYPE                                          VARCHAR2(200)
 START_NODE_ID                             NOT NULL NUMBER
 END_NODE_ID                               NOT NULL NUMBER
 COST                                               NUMBER
 SIMPLE                                             VARCHAR2(1)
 GEOMETRY                                           MDSYS.SDO_GEOMETRY

-- US_ROADS_PLINK table
```

```
SQL> describe US_ROADS_PLINK$
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PATH_ID                                   NOT NULL NUMBER
 LINK_ID                                   NOT NULL NUMBER
 SEQ_NO                                    NOT NULL NUMBER
```

The follow example illustrates the creation of the same US_ROADS network with explicit table and column names.

Network Creation with Explicit Table and Column Names:

```
SQL> BEGIN
  2     SDO_NET.CREATE_SDO_NETWORK (
  3     NETWORK                       => 'US_ROADS',
  4     NO_OF_HIERARCHY_LEVELS   => 1,
  5     IS_DIRECTED                  => TRUE,
  6     NODE_TABLE_NAME              => 'US_INTERSECTIONS',
  7     NODE_GEOM_COLUMN             => 'LOCATION',
  8     NODE_COST_COLUMN             => NULL,
  9     LINK_TABLE_NAME              => 'US_STREETS',
  10    LINK_GEOM_COLUMN             => 'STREET_GEOM',
  11    LINK_COST_COLUMN             => 'STREET_LENGTH',
  12    PATH_TABLE_NAME              => 'US_PATHS',
  13    PATH_GEOM_COLUMN             => 'PATH_GEOM',
  14    PATH_LINK_TABLE_NAME      => 'US_PATH_LINKS'
  15    );
  16    END;
  17    /

Network created.
```

- **"Manual" Network Definition**

When creating the network tables manually, it provides total flexibility over the table structures, but the network metadata must to be updated manually to ensure that the table structures are consistent with the metadata.

Below is an example of creating the US_ROADS spatial network manually with only the columns we need. Particularly, the US_ROADS network would be defined using street_length for the cost column that is contained in the us_streets table. It would allowed to calculate the shortest routes between nodes

Manual Network Creation:

```
-- Create the node table
SQL> CREATE TABLE us_intersections (
  2     node_id             NUMBER,
  3     location            SDO_GEOMETRY,
  4     CONSTRAINT us_intersections_pk PRIMARY KEY (node_id)
```

```
  5  );

Table created.

-- Create the link table
SQL> CREATE TABLE us_streets (
  2      link_id             NUMBER,
  3      start_node_id       NUMBER NOT NULL,
  4      end_node_id         NUMBER NOT NULL,
  5      active              CHAR(1),
  6      street_geom         SDO_GEOMETRY,
  7      street_length       NUMBER,
  8      travel_time         NUMBER,
  9      bidirected          CHAR(1),
 10      CONSTRAINT us_streets_pk PRIMARY KEY (link_id)
 11  );

Table created.

-- Create path table
SQL> CREATE TABLE us_paths (
  2      path_id             NUMBER,
  3      start_node_id       NUMBER NOT NULL,
  4      end_node_id         NUMBER NOT NULL,
  5      cost                NUMBER,
  6      simple              VARCHAR2(1),
  7      path_geom           SDO_GEOMETRY,
  8      CONSTRAINT us_paths_pk PRIMARY KEY (path_id)
  9  );

Table created.

-- Create path link table
SQL> CREATE TABLE us_path_links (
  2      path_id             NUMBER,
  3      link_id             NUMBER,
  4      seq_no              NUMBER,
  5      CONSTRAINT us_path_links_pk PRIMARY KEY (path_id, link_id)
  6  );

Table created.
```

Following example illustrates manually inserting the information into the network metadata (USER_SDO_NETWORK_METADATA).

Setting Up Network Metadata:

```
SQL> INSERT INTO USER_SDO_NETWORK_METADATA (
  2    NETWORK,
  3    NETWORK_CATEGORY,
  4    GEOMETRY_TYPE,
  5    NO_OF_HIERARCHY_LEVELS,
  6    NO_OF_PARTITIONS,
  7    LINK_DIRECTION,
  8    NODE_TABLE_NAME,
  9    NODE_GEOM_COLUMN,
  10    NODE_COST_COLUMN,
  11    LINK_TABLE_NAME,
  12    LINK_GEOM_COLUMN,
  13    LINK_COST_COLUMN,
  14    PATH_TABLE_NAME,
  15    PATH_GEOM_COLUMN,
  16    PATH_LINK_TABLE_NAME
  17    )
  18    VALUES (
  19    'US_ROADS',              -- network (primary key)
  20    'SPATIAL',              -- network_category
  21    'SDO_GEOMETRY',        -- geometry_type
  22    1,                          -- no_of_hierarchy_levels
  23    1,                          -- no_of_partitions
  24    'DIRECTED',             -- link_direction
  25    'US_INTERSECTIONS', -- node_table_name
  26    'LOCATION',             -- node_geom_column
  27    NULL,                       -- node_cost_column (no cost at node level)
  28    'US_STREETS',           -- link_table_name
  29    'STREET_GEOM',          -- link_geom_column
  30    'STREET_LENGTH',       -- link_cost_column
  31    'US_PATHS',             -- path_table_name
  32    'PATH_GEOM',            -- path_geom_column
  33    'US_PATH_LINKS'        -- path_link_table_name
  34    );

1 row created.

SQL> COMMIT;
```

- **Defining Multiple Networks on the Same Tables**

  the example in below shows the code of defining a second network on the us_streets table using travel_time as the cost column for the links to compute the fastest routes between nodes.

  Setting Up Metadata for a Time-Based Road Network:

```
SQL> INSERT INTO USER_SDO_NETWORK_METADATA (
  2     NETWORK,
  3     NETWORK_CATEGORY,
  4     GEOMETRY_TYPE,
  5     NO_OF_HIERARCHY_LEVELS,
  6     NO_OF_PARTITIONS,
  7     LINK_DIRECTION,
  8     NODE_TABLE_NAME,
  9     NODE_GEOM_COLUMN,
 10     NODE_COST_COLUMN,
 11     LINK_TABLE_NAME,
 12     LINK_GEOM_COLUMN,
 13     LINK_COST_COLUMN,
 14     PATH_TABLE_NAME,
 15     PATH_GEOM_COLUMN,
 16     PATH_LINK_TABLE_NAME
 17   )
 18   VALUES (
 19     'US_ROADS_TIME',       -- network (primary key)
 20     'SPATIAL',             -- network_category
 21     'SDO_GEOMETRY',        -- geometry_type
 22     1,                     -- no_of_hierarchy_levels
 23     1,                     -- no_of_partitions
 24     'DIRECTED',            -- link_direction
 25     'US_INTERSECTIONS', -- node_table_name
 26     'LOCATION',            -- node_geom_column
 27     NULL,                  -- node_cost_column (no cost at node level)
 28     'US_STREETS',          -- link_table_name
 29     'STREET_GEOM',         -- link_geom_column
 30     'TRAVEL_TIME',         -- link_cost_column
 31     'US_PATHS',            -- path_table_name
 32     'PATH_GEOM',           -- path_geom_column
 33     'US_PATH_LINKS'        -- path_link_table_name
 34   );

1 row created.

SQL> COMMIT;
```

Define a Network over Existing Structure

A simple approach of defining a network directly on the existing tables is to create views over the existing tables and rename columns in the views so that they match the naming conventions of the network tables, which include three steps, creating *views*, creating *path* and *path link table* and setting up the *network metadata*.

Below is an example of creating tables for a water distribution network, which is considered as an existing application.

Existing Water Network Tables:

```
SQL> CREATE TABLE valves (
  2     valve_id         NUMBER PRIMARY KEY,
  3     valve_type       VARCHAR2(20),
  4     location         SDO_GEOMETRY
  5     -- ... other columns ...
  6  );

Table created.

SQL> CREATE TABLE pipes (
  2     pipe_id          NUMBER PRIMARY KEY,
  3     diameter         NUMBER,
  4     length           NUMBER,
  5     start_valve      NUMBER NOT NULL REFERENCES valves,
  6     end_valve        NUMBER NOT NULL REFERENCES valves,
  7     pipe_geom        SDO_GEOMETRY
  8     -- ... other columns ...
  9  );

Table created.
```

The example steps show in below is used to define a network over existing structure:

1) The following example shows the code of defining views over the existing tables.

   Views over Existing Tables:

```
-- create net_valves view
SQL> CREATE VIEW net_valves (node_id, valve_type, location) AS
  2     SELECT valve_id,
  3            valve_type,
  4            location
  5            -- ...other columns ...
  6     FROM valves;

View created.

-- create net_pipes view
```

```
SQL> CREATE VIEW net_pipes (link_id, start_node_id, end_node_id, length,
pipe_geom) AS
  2     SELECT pipe_id,
  3               start_valve,
  4               end_valve,
  5               length,
  6               pipe_geom
  7               -- ... other columns ...
  8     FROM pipes;

View created.
```

2) The following example shows the creation of path and path link tables that is an optional step. They are used to keep the results of any network traces or analyses.

Creating the Path and Path Link Tables:

```
SQL> CREATE TABLE net_paths (
  2     path_id             NUMBER,
  3     start_node_id     NUMBER NOT NULL,
  4     end_node_id        NUMBER NOT NULL,
  5     cost                NUMBER,
  6     simple              VARCHAR2(1),
  7     path_geom           SDO_GEOMETRY,
  8     CONSTRAINT net_paths_pk PRIMARY KEY (path_id)
  9  );

Table created.

SQL> CREATE TABLE net_path_links (
  2     path_id             NUMBER,
  3     link_id             NUMBER,
  4     seq_no              NUMBER,
  5     CONSTRAINT net_path_links_pk PRIMARY KEY (path_id, link_id)
  6  );

Table created.
```

3) The following example shows the code of set up the network metadata for the water network.

Metadata for a Network on Existing Structures:

```
SQL> INSERT INTO USER_SDO_NETWORK_METADATA (
  2     NETWORK,
  3     NETWORK_CATEGORY,
  4     GEOMETRY_TYPE,
```

```
 5     NO_OF_HIERARCHY_LEVELS,
 6     NO_OF_PARTITIONS,
 7     LINK_DIRECTION,
 8     NODE_TABLE_NAME,
 9     NODE_GEOM_COLUMN,
10     NODE_COST_COLUMN,
11     LINK_TABLE_NAME,
12     LINK_GEOM_COLUMN,
13     LINK_COST_COLUMN,
14     PATH_TABLE_NAME,
15     PATH_GEOM_COLUMN,
16     PATH_LINK_TABLE_NAME
17   )
18   VALUES (
19     'WATER_NET',             -- network (primary key)
20     'SPATIAL',               -- network_category
21     'SDO_GEOMETRY',          -- geometry_type
22     1,                       -- no_of_hierarchy_levels
23     1,                       -- no_of_partitions
24     'UNDIRECTED',            -- link_direction
25     'NET_VALVES',            -- node_table_name
26     'LOCATION',              -- node_geom_column
27     NULL,                    -- node_cost_column (no cost at node level)
28     'NET_PIPES',             -- link_table_name
29     'PIPE_GEOM',             -- link_geom_column
30     'LENGTH',                -- link_cost_column
31     'NET_PATHS',             -- path_table_name
32     'PATH_GEOM',             -- path_geom_column
33     'NET_PATH_LINKS'         -- path_link_table_name
34   );

1 row created.

SQL> COMMIT;

Commit complete.
```

Validating Network Structure

The function that is used to verify the correctness of a network is VALIDATE_NETWORK function. It verifies the consistency between the metadata and the network tables, and the correction of tables that weather the right columns are with the right data types.

The example in below illustrate the validation of network US_ROADS.

Validating a Network Definition:

```
SQL> select sdo_net.validate_network('us_roads') from dual;

SDO_NET.VALIDATE_NETWORK('US_ROADS')
--------------------------------------------------------------------------------
  node geom metadata missing! link geom metadata missing! path geom metadata missing!
```

- **Dropping a Network**

  The DROP_NETWORK procedure is used to drop a network with any tables related to this network and the definition of the network from the metadata.

  Below is an example of removing the US_ROADS network.

  Dropping a Network:

```
SQL> EXEC SDO_NET.DROP_NETWORK ('US_ROADS');

PL/SQL procedure successfully completed.
```

- **Creating Spatial Indexes on Network Tables**

  Indeed, Spatial indexes need spatial metadata in the USER_SDO_GEOM_METADATA table. Below is an example of using the INSERT_GEOM_METADATA procedure to set the spatial metadata for all tables that are part of the network and that contain an SDO_GEOMETRY column.

  Adding Spatial Metadata to Network Tables:

```
SQL> BEGIN
  2     SDO_NET.INSERT_GEOM_METADATA (
  3       'US_ROADS',
  4       SDO_DIM_ARRAY (
  5          SDO_DIM_ELEMENT ('Long', -180, +180, 1),
  6          SDO_DIM_ELEMENT ('Lat', -90, +90, 1)
  7       ),
  8       8307
  9    );
 10  END;
 11  /

PL/SQL procedure successfully completed.

SQL> COMMIT;

Commit complete.
```

  In this example:

  - the function takes the name of the network as an input parameter: 'US_ROADS'

- **Getting Information About a Network**

  The details about the structure of a network can be displayed by querying the USER_SDO_NETWORK_METADATA view.

This example shows a procedure using functions to display information about a network.

Convenient Procedure for Getting Network Details:

```
SQL> CREATE OR REPLACE PROCEDURE SHOW_NET_DETAILS (NETWORK_NAME
VARCHAR2) AS
  2  BEGIN
  3     DBMS_OUTPUT.PUT_LINE ('Details on Network ' || NETWORK_NAME);
  4        DBMS_OUTPUT.PUT_LINE ('NETWORK_EXISTS() =                ' ||
SDO_NET.NETWORK_EXISTS(NETWORK_NAM
E));
  5        DBMS_OUTPUT.PUT_LINE ('IS_HIERARCHICAL() =               ' ||
SDO_NET.IS_HIERARCHICAL(NETWORK_NA
ME));
  6        DBMS_OUTPUT.PUT_LINE ('IS_LOGICAL() =                    ' ||
SDO_NET.IS_LOGICAL(NETWORK_NAME));

  7        DBMS_OUTPUT.PUT_LINE ('IS_SPATIAL() =                    ' ||
SDO_NET.IS_SPATIAL(NETWORK_NAME));

  8        DBMS_OUTPUT.PUT_LINE ('GET_NETWORK_CATEGORY() =          ' ||
SDO_NET.GET_NETWORK_CATEGORY(NETWO
RK_NAME));
  9        DBMS_OUTPUT.PUT_LINE ('SDO_GEOMETRY_NETWORK() =          ' ||
SDO_NET.SDO_GEOMETRY_NETWORK(NETWO
RK_NAME));
 10        DBMS_OUTPUT.PUT_LINE ('GET_NETWORK_TYPE() =              ' ||
SDO_NET.GET_NETWORK_TYPE(NETWORK_N
AME));
 11        DBMS_OUTPUT.PUT_LINE ('GET_GEOMETRY_TYPE() =             ' ||
SDO_NET.GET_GEOMETRY_TYPE(NETWORK_
NAME));
 12          DBMS_OUTPUT.PUT_LINE ('GET_NO_OF_HIERARCHY_LEVELS() = ' ||
SDO_NET.GET_NO_OF_HIERARCHY_LEVELS
(NETWORK_NAME));
 13         DBMS_OUTPUT.PUT_LINE ('GET_LINK_DIRECTION() =            ' ||
SDO_NET.GET_LINK_DIRECTION(NETWORK
_NAME));
 14        DBMS_OUTPUT.PUT_LINE ('GET_NODE_TABLE_NAME() =           ' ||
SDO_NET.GET_NODE_TABLE_NAME(NETWOR
K_NAME));
 15        DBMS_OUTPUT.PUT_LINE ('GET_NODE_COST_COLUMN() =          ' ||
SDO_NET.GET_NODE_COST_COLUMN(NETWO
RK_NAME));
 16        DBMS_OUTPUT.PUT_LINE ('GET_NODE_GEOM_COLUMN() =          ' ||
```

```
SDO_NET.GET_NODE_GEOM_COLUMN(NETWO
RK_NAME));
 17          DBMS_OUTPUT.PUT_LINE   ('GET_LINK_TABLE_NAME()   =                  '   ||
SDO_NET.GET_LINK_TABLE_NAME(NETWOR
K_NAME));
 18          DBMS_OUTPUT.PUT_LINE   ('GET_LINK_COST_COLUMN()   =               '   ||
SDO_NET.GET_LINK_COST_COLUMN(NETWO
RK_NAME));
 19          DBMS_OUTPUT.PUT_LINE   ('GET_LINK_GEOM_COLUMN()   =               '   ||
SDO_NET.GET_LINK_GEOM_COLUMN(NETWO
RK_NAME));
 20          DBMS_OUTPUT.PUT_LINE   ('GET_PATH_TABLE_NAME()   =                '   ||
SDO_NET.GET_PATH_TABLE_NAME(NETWOR
K_NAME));
 21          DBMS_OUTPUT.PUT_LINE   ('GET_PATH_GEOM_COLUMN()   =               '   ||
SDO_NET.GET_PATH_GEOM_COLUMN(NETWO
RK_NAME));
 22           DBMS_OUTPUT.PUT_LINE   ('GET_PATH_LINK_TABLE_NAME()   =          '   ||
SDO_NET.GET_PATH_LINK_TABLE_NAME(N
ETWORK_NAME));
 23  END;
 24  /

Procedure created.
```

- **Example Network**

  In this part, two sample networks, UNET and DNET, are created to illustrate network analysis functions.

UNET: A Simple Undirected Network

Below example shows the creation of a simple network (UNET).

Defining the UNET Network:

```
SQL> BEGIN
  2     SDO_NET.CREATE_SDO_NETWORK (
  3        NETWORK                          => 'UNET',
  4        NO_OF_HIERARCHY_LEVELS       => 1,
  5        IS_DIRECTED                     => FALSE,
  6        NODE_TABLE_NAME                 => 'UNET_NODES',
  7        NODE_GEOM_COLUMN                 => 'GEOM',
  8        NODE_COST_COLUMN                 => NULL,
  9        LINK_TABLE_NAME                 => 'UNET_LINKS',
 10        LINK_COST_COLUMN                 => 'COST',
 11        LINK_GEOM_COLUMN                 => 'GEOM',
 12        PATH_TABLE_NAME                 => 'UNET_PATHS',
```

```
 13        PATH_GEOM_COLUMN              => 'GEOM',
 14        PATH_LINK_TABLE_NAME          => 'UNET_PLINKS'
 15      );
 16    END;
 17    /

PL/SQL procedure successfully completed.
```

Below example shows the loading of UNET.

Loading the UNET Network:

```
-- Populate the node table
SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (1, 'N1',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (1,3,NULL), null, null));

1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (2, 'N2',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (2,3,NULL), null, null));

1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (3, 'N3',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (4,3,NULL), null, null));

1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (4, 'N4',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (2,2,NULL), null, null));

1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (5, 'N5',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (3,2,NULL), null, null));

1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (6, 'N6',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (4,2,NULL), null, null));
```

```
1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (7, 'N7',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (1,1,NULL), null, null));

1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (8, 'N8',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (3,1,NULL), null, null));

1 row created.

SQL> INSERT INTO unet_nodes (node_id, node_name, geom) VALUES (9, 'N9',
SDO_GEOMETRY (2001, NULL, SD
O_POINT_TYPE (4,1,NULL), null, null));

1 row created.

SQL> COMMIT;

Commit complete.

-- Populate the link table
SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 1, 'L1',    1, 2, 1, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (1,3, 2,3)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 2, 'L2',    2, 3, 2, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (2,3, 4,3)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 3, 'L3',    3, 6, 1, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (4,3, 4,2)));
```

```
1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 4, 'L4',  6, 9, 1, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (4,2, 4,1)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 5, 'L5',  9, 8, 1, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (4,1, 3,1)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 6, 'L6',  8, 7, 2, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (3,1, 1,1)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 7, 'L7',  7, 1, 2, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (1,1, 1,3)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 8, 'L8',  7, 4, 1.5, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SD
O_ORDINATE_ARRAY (1,1, 2,2)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2       VALUES ( 9, 'L9',  4, 5, 1, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (2,2, 3,2)));

1 row created.
```

```
SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2          VALUES (10, 'L10', 5, 6, 1, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (3,2, 4,2)));

1 row created.

SQL> INSERT INTO unet_links (link_id, link_name, start_node_id, end_node_id, cost, geom)
  2          VALUES (11, 'L11', 5, 8, 1, SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), SDO_
ORDINATE_ARRAY (3,2, 3,1)));

1 row created.

SQL> COMMIT;

Commit complete.
```

DNET: A Simple Directed Network

Below example shows the creation of a simple directed network (DNET).

Defining the DNET Network:

```
SQL> BEGIN
  2     SDO_NET.CREATE_SDO_NETWORK (
  3        NETWORK                          => 'DNET',
  4        NO_OF_HIERARCHY_LEVELS       => 1,
  5        IS_DIRECTED                     => TRUE,
  6        NODE_TABLE_NAME                => 'DNET_NODES',
  7        NODE_GEOM_COLUMN               => 'GEOM',
  8        NODE_COST_COLUMN               => NULL,
  9        LINK_TABLE_NAME                => 'DNET_LINKS',
 10        LINK_COST_COLUMN               => 'COST',
 11        LINK_GEOM_COLUMN               => 'GEOM',
 12        PATH_TABLE_NAME                => 'DNET_PATHS',
 13        PATH_GEOM_COLUMN               => 'GEOM',
 14        PATH_LINK_TABLE_NAME         => 'DNET_PLINKS'
 15     );
 16  END;
 17  /

PL/SQL procedure successfully completed.
```

Below example shows the code of loading the DNET.

Loading the DNET Network:

```
-- Populate the node table
SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2      VALUES (1, 'N1', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (1,3,NULL),
NULL, NULL));

1 row created.

SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (2, 'N2', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (2,3,NULL),
NULL, NULL));

1 row created.

SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (3, 'N3', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (4,3,NULL),
NULL, NULL));

1 row created.

SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (4, 'N4', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (2,2,NULL),
NULL, NULL));

1 row created.

SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (5, 'N5', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (3,2,NULL),
NULL, NULL));

1 row created.

SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (6, 'N6', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (4,2,NULL),
NULL, NULL));

1 row created.

SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (7, 'N7', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (1,1,NULL),
NULL, NULL));

1 row created.
```

```
SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (8, 'N8', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (3,1,NULL),
NULL, NULL));

1 row created.

SQL> INSERT INTO dnet_nodes (node_id, node_name, geom)
  2     VALUES (9, 'N9', SDO_GEOMETRY (2001, NULL, SDO_POINT_TYPE (4,1,NULL),
NULL, NULL));

1 row created.

SQL> COMMIT;

Commit complete.

-- Populate the link table
SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2     VALUES ( 1, 'L1',    2, 1, 1,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (2,3, 1,3)), 'N');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2     VALUES ( 2, 'L2',    3, 2, 2,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (4,3, 2,3)), 'N');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2     VALUES ( 3, 'L3',    3, 6, 1,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (4,3, 4,2)), 'Y');

1 row created.
```

```
SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES ( 4, 'L4',    6, 9, 1,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (4,2, 4,1)), 'Y');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES ( 5, 'L5',    9, 8, 1,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (4,1, 3,1)), 'Y');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES ( 6, 'L6',    8, 7, 3,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (3,1, 1,1)), 'N');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES (-6, 'L6',    7, 8, 2,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (1,1, 3,1)), 'N');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES ( 7, 'L7',    1, 7, 2,       SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (1,3, 1,1)), 'N');
```

```
1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES ( 8, 'L8',    4, 7, 1.5,    SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (2,2, 1,1)), 'N');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES ( 9, 'L9',    5, 4, 1,      SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (3,2, 2,2)), 'N');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES (10, 'L10', 5, 6, 1,      SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (3,2, 4,2)), 'N');

1 row created.

SQL> INSERT INTO dnet_links (link_id, link_name, start_node_id, end_node_id, cost, geom,
bidirected)

  2      VALUES (11, 'L11', 8, 5, 1,      SDO_GEOMETRY (2002, NULL, NULL,
SDO_ELEM_INFO_ARRAY (1,2,1), S
DO_ORDINATE_ARRAY (3,1, 3,2)), 'N');

1 row created.

SQL> COMMIT;

Commit complete.
```

11.0 CONCLUSION

Oracle Spatial is efficient tool for managing and exploiting spatial information, which provide data analysis in business applications for business operations and decision-making. In addition, this report has adopted real case examples to execute the experiments on spatial analysis using SQL*Plus, although SQL API and Java can be used for the same purpose. Indeed, the project carried out with some difficulties, although the majority of the tasks were completed. The concepts of using SQL queries to generate geographical analysis by executing spatial data in Oracle Spatial are built during this project. Moreover, many different methods of performing the spatial data analysis was learnt thus this project, and most of the important features and functions have been examined successfully, which are covered in this paper. However, there are also trials and errors that occurred during the execution, which is mainly related to the Spatial Indexes.

Some parts of the experiments were unable to be completed to the end, because the issue with Oracle MapViewer, which is runs separate from SQL*plus. Therefore, the parts in defending MapViewer and using MapViewer in business application were impossible to be finished. Furthermore, a few parts in the spatial network modeling and others such as routing engine could also not be examined which considered about the availability of service.

REFERENCE

- Ravi Kothuri, Albert Godfrind, and Euro Beinat, (2007). “Pro Oracle Spatial for Oracle Database 11g”, Springer-Verlag New York, Inc., USA