# Oracle Spatial

## Best Practices

*An Oracle Technical White Paper*
*December 2003*

## Table Of Contents

Oracle Spatial and Oracle Locator are powerful core features of the Oracle database. This technical document describes some best practices, tips and general information that can help utilize Oracle Spatial and Oracle Locator to increase productivity, decision support, and cost savings in your everyday business practices.

Oracle Spatial in Oracle Database10*g* includes native data types for storing vector, raster, and persistent topology data. This document outlines some best practices when working with Oracle's native vector data type, SDO_GEOMETRY, in Oracle Spatial for Oracle Database 9*i* and 10*g*.

## 1 Overview

Oracle Locator is bundled with Oracle's Standard and Enterprise editions. Locator is a subset of Spatial. Licensing Oracle's Standard or Enterprise editions enables you to fully leverage the valuable set of features offered in Oracle Locator, at no extra cost. SDO_GEOMETRY is the only vector data type available in Oracle, natively integrated into the product suites of all the major GIS vendors.

Some of the major GIS vendors have legacy proprietary alternatives to store vector data in Oracle, for example, in Oracle's LONG RAW data type. These legacy alternatives were introduced prior to the availability of the SDO_GEOMETRY data type. In the marketplace, there is sometimes the misconception that using SDO_GEOMETRY as the underlying vector data store may compromise features or performance in products offered by the major GIS vendors. The truth is that the SDO_GEOMETRY data type does not compromise any of the superb features offered by major GIS vendor products. If you follow the guidelines described in this document, the best alternative proprietary data stores should never deviate more than +/- 15% of the performance offered by the SDO_GEOMETRY data type.

Indeed, if you leverage core Oracle features like table partitioning, Oracle Spatial performance may far exceed 15% better performance over proprietary "LONG RAW" data type solutions. This is because Oracle does not support table partitioning on tables that contain LONG or LONG RAW columns. Tables with SDO_GEOMETRY columns can leverage Oracle table partitioning, which can significantly help performance, scalability, and manageability.

Oracle Spatial was designed to leverage core features of the Oracle database. By leveraging core utilities and features of Oracle, existing Oracle DBAs and developers can maximize their existing Oracle knowledge base when working with Oracle's spatial technologies. If you know Oracle, and never heard of Oracle Spatial, you already know over 80% of this Oracle feature. The same core Oracle utilities (import, export, sqlldr) used for non-spatial data, are also used with spatial data. Similarly, core Oracle features such as table partitioning and advanced replication all work with Oracle Spatial. Oracle's spatial strategy is to mainstream spatial data in your organization and maximize the existing Oracle knowledge of your enterprise.

This paper highlights some best practices and tips to help design and develop applications that leverage Oracle's spatial technology. Many of the recommendations in this paper are not Oracle Spatial specific, further emphasizing the ability to leverage existing Oracle knowledge within your enterprise. The rest of this paper contains the following sections:

- Data Modeling
- Metadata, Tolerance, and Coordinate Systems
- Data Loading
- Data Validation
- Indexing Spatial Data
- Partitioned Spatial Indexes
- Spatial Queries
- Application Considerations

## *2 Data Modeling*

Traditional RDBMS data model concepts apply when dealing with spatial data. Oracle supports many traditional data types, including VARCHAR2 for characters, DATE type for dates, NUMBER type for numbers, and now an SDO_GEOMETRY data type for storing the coordinates of spatial features.

There is no such thing in Oracle as a spatial table, just ordinary Oracle tables with one or more SDO_GEOMETY columns. When you create normalized tables, Oracle recommends including SDO_GEOMETRY columns in tables where all the other columns in the table have a one-to-one relationship with the SDO_GEOMETRY column.

Consider the following example of modeling road and river spatial features. Road information might include number of lanes, a street address range, and more. River information might include salinity, maximum depth, and more. Even though they are both linear features, since the information for roads is not relevant to rivers, and river information is not relevant to roads, it is not recommended to store their coordinates in the same SDO_GEOMETRY column of a table. A normalized data model would store the road spatial features in a Roads table, along with other columns that have a one to one relationship with the coordinates of a road. A similar normalized data model is recommended for a Rivers table.

An additional benefit of storing roads apart from rivers becomes more apparent at query time. When you are only searching for roads, there is no need to sift through a table that contains entries for both roads and rivers.

## *3 Metadata, Tolerance, and Coordinate Systems*

Every SDO_GEOMETRY column in a table requires an entry in Oracle spatial metadata dictionary, USER_SDO_GEOM_METADATA. The metadata entry includes the following information:

- Name of the table that contains the column of type SDO_GEOMETRY
- Name of the column defined with the SDO_GEOMETRY data type
- Number of axes (dimensions) for the SDO_GEOMETRY column
- Lower and upper bounds for each axis
- Tolerance value for each axis, generally the same value for all axes
- Spatial reference identifier (SRID)

The lower and upper bound of each axis is not the minimum bounding rectangle (MBR) of the data in the SDO_GEOMETRY column. The axes bounds should be values that contain all current and future geometries. The first axis defined must always be x, and the second axis y. Optional z and measure axes can also be defined.

When dealing with geodetic data (data that is longitude/latitude), the first axis must be defined with a (-180, 180) range, and the second axis as (-90,90).

Tolerance is generally the same for both the x and y axes. Tolerance is the distance two coordinates must be apart to be considered unique. Oracle's geometry validation routines, spatial operators, and spatial functions all use tolerance. It is very important to define a tolerance that reflects the true resolution at which your data was collected.

When storing data that is not longitude/latitude, the tolerance unit is the same as the coordinate system unit associated with the spatial data. When storing longitude/latitude data, the tolerance unit is meters.

All coordinate systems supported by Oracle Spatial and Oracle Locator are defined in a dictionary table called MDSYS.CS_SRS. Custom coordinate systems can also be added to the MDSYS.CS_SRS dictionary, and the process is described in the *Oracle Spatial Users Guide and Reference*. In the MDSYS.CS_SRS dictionary, a numeric primary key called the SRID identifies each supported coordinate system. The dictionary table also contains the definition of each coordinate system in the well known text (WKT) grammar defined by the Open GIS Consortium (OGC). Associating spatial data with a coordinate system is as simple as associating the spatial data with an SRID value.

Associating spatial data with an SRID is recommended, especially if your data is geographic, that is, related to the Earth. Geographic data can be divided into two categories, geodetic (longitude/latitude data), and projected (non-longitude/latitude data). Oracle considers Great Circle distances between consecutive coordinates of geometries defined with a geodetic SRID.

When associating an SRID with an SDO_GEOMETRY column, it must be specified in the USER_SDO_GEOM_METADATA entry, and also in the SDO_SRID attribute of each SDO_GEOMETRY object loaded.

## *4 Data Loading*

Bulk loads can be accomplished with traditional Oracle utilities, such as SQL*Loader and Import. Bulk unloading can be accomplished with Oracle's Export utility. These utilities require no spatial specific syntax. As recommended with non-spatial data, if you are performing a large bulk load, it is recommended to drop indexes (including spatial indexes if they exist), perform the load, and recreate indexes after the load completes. If indexes are not dropped prior to a bulk load, they are maintained as the load occurs.

SQL*Loader can load spatial data, but it does not understand Geographic Information System (GIS) vendor exchange formats, such as ESRI shapefiles, MapInfo Tab files, Autodesk DWG files, or Microstation DGN files. Each major GIS vendor has their own tool to import their exchange formats into Oracle's SDO_GEOMETRY format. There are also universal translation products, such as Feature Manipulation Engine (FME) by Safe Software, that can load numerous vendor formats into the SDO_GEOMETRY data type. FME can also extract data stored in Oracle's SDO_GEOMETRY data type and translate it to any of FME's supported GIS vendor formats.

Since ESRI shapefiles are a very common exchange format, Oracle posts a free utility, shp2sdo, on the Oracle Technology Network. You can download this utility from the following URL: http://otn.oracle.com/products/spatial. The utility is unsupported by Oracle, but is an excellent tool that has been tested extensively. The tool is unsupported because it is coded with shareware to read shapefiles. The shp2sdo utility reads geometries and attributes from a shape file. It then creates:

- A SQL script to create the corresponding Oracle table, and associated metadata required for the SDO_GEOMETRY column
- A control file for the Oracle Import utility

If you are using Oracle's shp2sdo utility, after data load it is recommended that you run the SDO_MIGRATE.TO_CURRENT procedure to migrate the data to the latest format recommended by Oracle. If you are using a GIS vendor utility, or Safe Software's FME product, it is not necessary to run SDO_MIGRATE.TO_CURRENT.

## *5 Data Validation*

Spatial data must be valid to ensure correct results when you perform spatial analysis. If an SDO_GEOMETRY column is spatially indexed, Oracle will perform some validity checks when spatial data is inserted into the column. But complete validation only occurs by running either the SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT or SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT procedure.

If data is guaranteed to be valid prior to data load, validation is not necessary. Otherwise, validation is highly recommended. Invalid geometries should either be corrected or deleted.

SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT and SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT validate geometries in accordance with rules defined by the Open GIS Consortium (OGC) via the Simple Feature Specification for SQL. When invalid geometries are reported (for example, a self-intersecting polygon), additional context information, such as which edges intersected, is also reported. The additional context information is very useful in correcting invalid geometries.

Some of the most common validation errors reported include:

| **Error Reported** | **Possible Causes and Corrective Actions** |
|---|---|
| ORA 13356 – Adjacent repeated points in a geometry are redundant. | Tolerance may be set too coarsely. Making tolerance finer may fix the error.<br><br>Points may truly be repeated. Remove duplicate vertices. |
| ORA 13349 – Polygon boundary crosses itself. | Tolerance may be set too coarsely. Making the tolerance finer may fix the error.<br><br>Polygon truly self intersects. Fix the polygon by ensuring that no edges intersect. |
| ORA 13367 – Wrong rotation for interior/exterior rings. | Correct the rotation of the polygon ring. Outer rings should be counterclockwise, inner rings clockwise. |

The additional context information reported by the validation routines can be supplied to the following routines to help fix invalid geometries:

- SDO_UTIL.REMOVE_DUPLICATE_VERTICIES
- SDO_UTIL.EXTRACT

## 6 *Indexing Spatial Data*

R-Tree spatial indexes, introduced in Oracle 8.1.7, require no tuning, and are recommended in almost all scenarios. Oracle9*i* introduced a geodetic R-tree index, which takes into account Great Circle distances, and also geometries that span the poles and the 180 meridian. Oracle9*i* Release 2 introduced some parallelization when creating R-tree spatial indexes, and also performance enhancements. Oracle Database 10*g* includes more R-tree performance enhancements, concurrent DML enhancements, and more parallelization.

Specifying the LAYER_GTYPE parameter in the CREATE INDEX statement will:

- Help maintain spatial integrity, by only allowing a certain class of spatial features to be inserted in to the SDO_GEOMETRY column. For example, specifying LAYER_GTYPE=POINT at index creation will allow only point data to be inserted into the column.
- Help query performance. Spatial queries optimizations will be invoked if the class of spatial data for the column is specified at index creation.

The following CREATE INDEX parameters are not associated with R-tree spatial indexes, and should never be specified:

- SDO_LEVEL
- SDO_NUMTILES
- SDO_COMMIT_INTERVAL

## *7 Partitioned Spatial Indexes*

Oracle table partitioning was introduced in Oracle8 to help improve performance and manageability of Oracle tables with many rows. Local partitioned spatial indexes were introduced in Oracle9*i*, and can provide much better performance and manageability for large Oracle tables that contain indexed SDO_GEOMETRY columns. For improved performance, pick a partition key that will appear in the WHERE clause most of the time.

If you are designing a very large database system, and possibly considering a GIS vendor's legacy proprietary LONG RAW spatial data store, you may want to think twice. Oracle does not support partitioning tables that contain any LONG or LONG RAW columns. These solutions will definitely begin to degrade in performance and manageability as tables get larger.

The Oracle Spatial SDO_GEOMETRY data type can leverage the benefits of local spatial partitioned indexes for faster, localized searches, and also manageability features such as partition level backups and exchanges.

With partition-level exchange, an entire table that resides outside of a partitioned table can be swapped into the partitioned table, as a partition. All of the indexes (including spatial indexes) that existed on the table outside the warehouse immediately become available in the partitioned table. This is a very quick way to add new data to a warehouse, minimizing index maintenance.

## *8 Spatial Queries*

Oracle Spatial includes a set of spatial operators and functions. Spatial operators and functions both perform spatial analysis, the difference being operators leverage spatial indexes and functions do not. Spatial operators include the following:

- SDO_FILTER (search_column, window, 'operator specific parameters')
- SDO_RELATE (search_column, window, 'operator specific parameters')
- SDO_WITHIN_DISTANCE (search_column, window, 'operator specific parameters')
- SDO_NN (search_column, window, 'operator specific parameters')
- And others introduced in Spatial for Oracle Database 10*g*

Spatial functions include the following:

- SDO_GEOM.SDO_AREA

- SDO_GEOM.SDO_LENGTH
- SDO_CS.TRANSFORM
- SDO_LRS.PROJECT_PT
- And many others

Notice that all spatial operators have a similar parameter signature:

- The first argument of a spatial operator is always the column being searched, and must be spatially indexed.
- The second argument is always the query window, or area of interest.
- The third argument is a string that includes a list of parameters specific to that operator.

When you are writing SQL statements that include spatial operators, Oracle hints can help the Oracle optimizer choose a better execution plan. Oracle hints are not Spatial-specific, but just as they can improve execution plans for SQL statements that do not include spatial operators, they can also help execution plans for SQL statements that do.

For an optimal execution plan, always specify the /*+ ordered */ hint when multiple geometries are being passed into the second argument of a spatial operator. For example, the following query finds all the chemical plants within 5 miles of contaminated wells with ID values 1 and 2.

```
SELECT /*+ ORDERED */
        b.chemical_plant_name
FROM well_table a,
      chemical_plants b
WHERE sdo_within_distance (b.geom, a.geom, 'distance=5 unit=mile') =
       'TRUE'
  AND a.id in (1,2);
```

In the example, multiple well locations (well IDs 1 and 2) are passed into the second argument of the operator. This is a classic example where the /*+ ordered */ hint should be specified. When you specify the ordered hint, list the tables in the FROM clause in driving table order. The table that feeds the second argument of the spatial operator should be listed in the FROM clause before the table that contains the search column. In this example, well_table is listed before chemical_plants in the FROM clause.

Spatial operators narrow the result candidates by excluding rows from the table associated with the first argument of the operator (the search column). When the results of a query are narrowed by a spatial operator, and when other indexed columns from the same table that feeds argument 1 of the spatial operator appear in the WHERE clause, it may be helpful to use a no_index hint. This is especially true if those other indexed columns are not very selective.

For example, assume the chemical plant query was modified to return all the chemical plants "that process chromium", within 5 miles of contaminated wells with ID values 1 and 2. Assume that the chemical_plants table has a column called processes_chromium, with possible values of ‘T’ or ‘F’ (true or false). Even if the processes_chromium column has a bitmap index on it, it probably would not be very selective index to use in the query. Providing a no_index hint on processes_chromium index can help the Oracle optimizer avoid a merge of the selective spatial index and the non-selective, non-spatial index.

```
SELECT /*+ ORDERED
     NO_INDEX (b processes_chromium_index_name) */
     b.chemical_plant_name
FROM well_table a,
     chemical_plants b
WHERE sdo_within_distance (b.geom, a.geom, 'distance=5 unit=mile') =
     'TRUE'
  AND a.id in (1,2)
  AND processes_chromium = 'T';
```

Oracle recommends cost based optimization, and gathering statistics on tables and indexes. This can be accomplished with the following procedures:

- DBMS_STATS.GATHER_TABLE_STATISTICS
- DBMS_STATS.GATHER_SCHEMA_STATISTICS

The table that contains the SDO_GEOMETRY data type can benefit by gathering statistics on it, but it is not necessary to gather statistics on the spatial index table implicitly created by the CREATE INDEX command.

## 9 Application Considerations

If visualization is a key component of your application, this section may be very relevant. When the display is zoomed out very far, it is not good practice to turn on very detailed layers. For example, if the display is zoomed out to show the entire United States, turning on detailed streets does not add value to the display. Detailed streets at that zoom level would appear as a solid blob on the screen. It is much more realistic to turn on detailed streets when the display is zoomed in to a one kilometer area east to west.

The following is another example. Assume that you have a layer with very detailed polygon regions (about 3000 vertices in each polygon). When you are zoomed out very far, it does not make sense to display these very detailed polygons. The detail of the polygons is lost because the many coordinates in these polygons are being forced to render onto just a few pixels. A more realistic scenario would be to use zoom control to only turn on the detailed polygons when you are reasonably zoomed in. Another realistic approach is to create a generalized layer (generalized version of the detailed polygons). The generalized layer is displayed when you are zoomed out very far, and the detailed layer is displayed when you are zoomed closer in.

Zoom control and the use of generalized layers are very well known concepts for display applications. Correct usage of zoom control and generalized layers will provide much better performance. Unnecessary fetches of detailed geometries from the server can be avoided, especially since most of the coordinates of each detailed geometry would render on just a few pixels.

Oracle Spatial Best Practices
Author: Daniel Geringer