

Oracle Locator and Oracle Spatial 11g Best Practices

*An Oracle Technical White Paper
August 2008*

Oracle Locator and Oracle Spatial 11g Best Practices

- Introduction 3
- General Best Practices 4
 - Oracle patchsets 4
 - Data modeling 4
 - Metadata, tolerance and coordinate systems 4
 - Transportable tablespace with spatial data and spatial indexes 6
 - Data loading 6
 - Point data in the SDO_POINT attribute of the SDO_GEOMETRY object 6
 - Geometry validation 7
 - Spatial index creation 8
 - Partitioned local spatial indexes 9
 - Spatial Queries 9
 - Application considerations 11
- Best Practices with ESRI ArcGIS 12
 - DBTUNE file 13
 - Important notes for 3D data: 13
 - Creating a spatial index with an ESRI client tool 14
 - Primary key for ArcSDE 14
 - Registration of spatial layers not created by ESRI client tools 14
 - Deregistering layers from ArcSDE 14
 - Performance with ESRI client tools: 15

Oracle Locator and Oracle Spatial 11g Best Practices

INTRODUCTION

This technical white paper describes best practices for Oracle Locator and Oracle Spatial with `SDO_GEOMETRY`, the Oracle Database native data type for storing vector data. It also provides guidance for running ESRI ArcGIS software on Oracle Locator and Oracle Spatial.

Oracle Spatial extends the core location features included in every Oracle database with Oracle Locator. Therefore, this document refers to Oracle Locator when describing best practices that apply to both Oracle Locator and Oracle Spatial. It refers to Oracle Spatial when practices are specific to Oracle Spatial.

This paper highlights some best practices and tips to help design and develop applications that use Oracle spatial technology. Many of the recommendations in this paper are not specific to Oracle Locator, allowing customers to capitalize on existing Oracle knowledge within their enterprise.

Oracle Locator support by GIS and location tools and applications, and for standards: All major GIS and location services technology vendors, and mapping and imagery data providers directly integrate with Oracle Locator. They support the native Oracle `SDO_GEOMETRY` data type without compromising features or performance. Oracle also consistently works to help shape, drive, implement and support the latest open standards in the spatial and location services areas, as put forward by the Open Geospatial Consortium (OGC) and ISO SQL/MM standard.

Oracle Locator and Oracle Spatial are native features of Oracle Database providing security, high availability, disaster tolerance, manageability and performance for all your data, including your spatial data. Only users of the Oracle native spatial data type can take full advantage of all of these features: partitioning, local partitioned spatial indexes, partition exchange including spatial indexes, transportable table space support for spatial data and spatial indexes, replication, workspace manager (versioning, and valid time), parallel index builds and queries, and spatially-driven multi-level security. These features are not available or limited in functionality when using the `LONG RAW` or `BLOB` data types.

Oracle DBAs and developers capitalize on their existing Oracle knowledge since Oracle spatial technologies are integrated with core Oracle Database utilities and features listed in this and the previous paragraph. If you know Oracle Database, and never heard of Oracle Locator, you already know over 80% of this

Oracle feature. The same core Oracle utilities (impdp, expdp, import, export, sqldr) used for non-spatial data, are also used with spatial data. Oracle enables mainstreaming spatial data in your organization and capitalizing on existing Oracle knowledge in your enterprise.

GENERAL BEST PRACTICES

Oracle patchsets

If possible, upgrade to the latest Oracle Database patch set. This will ensure you are running the Oracle Database version with the latest spatial features and performance optimizations.

Data modeling

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

Oracle Database does not have spatial tables, just ordinary Oracle Database tables with one or more SDO_GEOMETRY 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.

Metadata, tolerance and coordinate systems

Every SDO_GEOMETRY column in a table requires an entry in the Oracle Locator 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-axes 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 Locator are defined in a dictionary table called `MDSYS . CS_SRS`. In addition, a data model for representing coordinate systems in the European Petroleum Survey Group (EPSG) standard was introduced in Oracle 10g Release 2. Every EPSG defined coordinate system has a corresponding entry in `MDSYS . CS_SRS`.

Oracle Locator comes pre configured with over 1000 coordinate systems defined. The pre-configured coordinate systems satisfy the requirement of many users. If none of the pre-configured coordinate systems meet your needs, you can add your own custom coordinate system.

Beginning with Oracle Database 10g release 2, custom coordinate systems must be entered via the EPSG data model defined in Oracle Database. Every custom EPSG entry will automatically generate an entry into the `MDSYS . CS_SRS` table. The process to add a custom coordinate system is described in the Oracle Spatial Users Guide and Reference.

In the `MDSYS . CS_SRS` dictionary, a numeric primary key called 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 geodesic distances between consecutive coordinates of geometries defined with a geodetic SRID.

Transportable tablespace with spatial data and spatial indexes

If moving large amounts of spatial data and spatial indexes from one database instance to another, you may want to consider transportable tablespaces.

Beginning with Oracle Database 10g, transportable tablespaces support transporting spatial indexes, as long as the source and target platforms are the same Endian format.

Transporting just the spatial data (not the spatial index) does not require the source and target platforms to have the same Endian format.

If you plan to transport tablespaces, you may want to reconsider how data and indexes are mapped to tablespaces to help optimize your transport strategy.

Data loading

Bulk loads can be accomplished with traditional Oracle Database utilities, such as `sqlldr`, `imp`, or `impdp`. Bulk unloading can be accomplished with utilities like Oracle Database export utilities, `exp` or `expdp`. These utilities require no spatial specific syntax. As recommended with non-spatial data, if you are performing a large bulk load or unload, it is recommended to drop indexes (including spatial indexes if they exist), perform the bulk transaction, and recreate indexes after the transaction completes. If indexes are not dropped prior to a bulk load or unload, they are maintained as the transaction 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 Java utility on the Oracle Technology Network for loading shapefiles. You can download this utility from the following URL:

<http://otn.oracle.com/products/spatial>. The utility is an excellent tool that has been tested extensively. It reads shapefile geometries along with their attributes, and loads (or appends) the records into an Oracle table.

Point data in the SDO_POINT attribute of the SDO_GEOMETRY object

When possible, store point data in the `SDO_POINT` attribute of the `SDO_GEOMETRY` object instead of the `SDO_ORDINATES` attribute.

Geometry 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. 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. 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`

The `SDO_UTIL` package contains other helpful utilities for inspecting geometries, for example:

- `SDO_UTIL.GETNUMELEM` – Returns the number of elements in a geometry
- `SDO_UTIL.GETNUMVERTICES` – Returns the number of vertices in a geometry
- `SDO_UTIL.GETNUMRINGS` – Returns the number of rings in a geometry.

See the documentation for additional utilities.

Spatial index creation

The following parameters are recommended when creating spatial indexes.

- `WORK_TABLESPACE` - During spatial index creation, the process creates intermediate tables that get dropped when the index is complete. The intermediate tables can take up to 2 times the size of the final index. If `WORK_TABLESPACE` is not specified, the intermediate tables are created in the same tablespace as the final index, causing fragmentation, and possible performance degradation. You can use `SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE`, and multiply the result by 2 to provide guidance on sizing the work tablespace. The work tablespace can be re-used to create other spatial indexes.
- `LAYER_GTYPE` – This parameter is needed especially when working with point-only layers. If a point-only layer stores its points in the `SDO_ORDINATE_ARRAY`, you can still specify `LAYER_GTYPE=POINT` on spatial index creation. This can help query performance when performing spatial analysis.
- `SDO_NON_LEAF_TBL` – This parameter is useful for very large spatial indexes (not necessary for smaller spatial indexes). This generates two spatial index tables instead of one. The smaller spatial index table is the non-leaf table, which is traversed most often during spatial analysis. It can be beneficial to pin the non-leaf table into the buffer pool, since it is accessed most often. See the example below.

-- Create the index

```
CREATE INDEX geod_counties_sidx ON geod_counties (geom)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS ('sdo_non_leaf_tbl=TRUE');
```

-- Find the non leaf index table name

```
SELECT sdo_nl_index_table
FROM user_sdo_index_metadata
WHERE sdo_index_name='GEOD_COUNTIES_SIDX';
-----
MDNT_A930$
```

-- Pin the table in memory

```
ALTER TABLE MDNT_A930$ STORAGE (BUFFER_POOL KEEP);
```

Partitioned local spatial indexes

The following approach is recommended to use for parallel creation of local partitioned spatial indexes on partitioned tables. With this approach, if any ALTER INDEX commands fail, you do not have to rebuild any local partitioned indexes that have already successfully completed.

- Create the LOCAL spatial index with the UNUSABLE keyword. This runs very quickly and only creates metadata associated with the index:

```
CREATE INDEX sp_idx ON my_table (location)
INDEXTYPE IS mdsys.spatial_index PARAMETERS
('tablespace=tb_name
work_tablespace=work_tb_name')
LOCAL UNSUABLE;
```

- Build individual scripts. Hypothetically, if you have 100 partitions and 10 processors, build 10 scripts, each with 10 ALTER INDEX REBUILD statements... but do not use the PARALLEL parameter. For example:
 - ALTER INDEX sp_idx REBUILD PARTITION ip1;
 - ALTER INDEX sp_idx REBUILD PARTITION ip2;
 - etc...
- Run all 10 scripts at the same time. Each processor will be working on a single partition's local index, but all the processors will still be busy working on their own set of ALTER INDEX statements.

Note: It is no longer recommended to use the PARALLEL keyword for parallel creation of spatial indexes on partitioned tables, the reason being that when creating a partitioned LOCAL spatial index, if any partitions index fails (for example, tablespace full or for some other reason), you must start at the beginning again (no way to continue where you left off).

Spatial Queries

Oracle Locator and Oracle Spatial include 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.

Operators include the following:

- SDO_FILTER (search_column, window)
- SDO_RELATE (search_column, window, 'operator specific parameters')
- SDO_ANYINTERACT (search_column, window)
- SDO_INSIDE (search_column, window)

- SDO_WITHIN_DISTANCE (search_column, window, 'operator specific parameters')
- SDO_NN (search_column, window, 'operator specific parameters')

See the documentation for many additional operators.

Functions include the following:

- SDO_GEOM.SDO_AREA
- SDO_GEOM.SDO_LENGTH
- SDO_CS.TRANSFORM
- SDO_LRS.PROJECT_PT

See the documentation for many additional functions.

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, if it exists, 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 Oracle Locator 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 the query window (second argument of a spatial operator) comes from a table. 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, well locations (well IDs 1 and 2) passed into the second argument of the operator come from the `well_table`. 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.

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 for each of these detailed geometries would render on just a few pixels.

BEST PRACTICES WITH ESRI ARCGIS

ESRI has supported Oracle Database for over 12 years with thousands of mutual customers. ESRI has been a beta site for every Oracle Database release since 1995.

ESRI's geodatabase is fully supported with Oracle Locator and Oracle Spatial, with the ArcGIS features and performance needed by real world applications. The substantial mutual customer base substantiates this. The performance was quantified most recently in the paper [Spatial Database Speed Comparisons](#), presented by Kevin S. Larson, Idea Integration, at the Geospatial Information Technology Association (GITA) 2008 Geospatial Infrastructure Solutions Conference.

The goal of the paper was to determine which database and data type performs best with ESRI ArcGIS. ESRI ArcGIS version 9.2 was tested with Oracle Database 10.2.0.3 using all the data types supported by ESRI SDE on Oracle Database. The study tested hot versus cold cache, a good representation of point, line and polygon data, and spatial data at varying densities as found in different geographic regions and data scales. Two sets of query workloads were performed: 1) Spatial data only queries involving only geometry data and 2) Business queries involving both geometry and relational data.

The study found that Oracle Locator with its native Oracle SDO_GEOMETRY data type provided the most consistent and scaleable performance for ESRI ArcGIS on Oracle Database. It provides significantly better performance for larger data sets (smaller data scales) and virtually same performance at lower data densities. The SDO_GEOMETRY performance advantage was more pronounced in spatial-only queries. These findings are consistent with those reported by customers.

What follows are some specific best practices for tuning ESRI ArcGIS with Oracle Locator as determined by customer experiences.

DBTUNE file

The following are recommended attributes to add to the DBTUNE keyword associated with your SDO_GEOMETRY feature classes. The values below are an example. Attribute values should be set with appropriate values for your SDO_GEOMETRY feature classes.

If your data has values per point, for example, (x,y) and a measure (xyz), make sure you add SDO_LB_3, SDO_UB_3, SDO_TOLERANCE_3 attributes to your DBTUNE keyword. If your data is 4 numbers per point, for example, (x,y,z) and a measure, make sure you add SDO_LB_4, SDO_UB_4, SDO_TOLERANCE_4 attributes to your DBTUNE keyword.

Important notes for 3D data:

- If your data is 3D, examine your geometries to see if the Z ordinate is always 0. This is sometimes the result of utilities that load CAD data into SDO_GEOMETRY. If you notice the Z ordinate is always 0, it is recommended to remove all the Z ordinates from your geometries. If the Z ordinate is not removed, 1/3 more geometry data is transferred over the network than necessary. In other words, transferring the Z ordinate consumes time; even if it's value is always 0.
- Utilities that load CAD data into SDO_GEOMETRY usually have a flag you can set to upload the data as 2D.

SDO_INDEX_SHAPE is where you can specify SDO_GEOMETRY CREATE INDEX parameters for the scenario where an ESRI client tool creates the spatial index for an SDO_GEOMETRY feature class.

```
GEOMETRY_STORAGE "SDO_GEOMETRY"  
SDO_DIMNAME_1    "x"  
SDO_DIMNAME_2    "y"  
SDO_LB_1         -180.000000  
SDO_LB_2         -90.000000  
SDO_SRID         8307  
SDO_TOLERANCE_1  0.05  
SDO_TOLERANCE_2  0.05  
SDO_UB_1         180.000000  
SDO_UB_2         90.000000  
SDO_VERIFY       "FALSE"  
SDO_INDEX_SHAPE  "tablespace=value  
WORK_TABLESPACE=value etc.."
```

The DBTUNE parameters are stored in the database. To extract the current DBTUNE file, you can issue the following command. It will place a file called dbtune_orig.out in SDEHOME\etc\dbtune.orig.

```
sdedbtune -o export -f dbtune_orig.out -u sde -p  
sde_password
```

```
copy dbtune_orig.out dbtune.out
```

Add the SDO_GEOMETRY specific DBTUNE parameter discussed above to dbtune.out. Then load the new dbtune.out file into the database as follows:

```
sdedbtune -o import -f dbtune.out -u sde -p sde
```

Creating a spatial index with an ESRI client tool

Please see the best practices for index creation. If an ESRI client tool loads data, there is usually an option to create or not create the spatial index. If you chose to have an ESRI client tool create the spatial index, make sure the DBTUNE attributes discussed in the DBTUNE section of this document are set appropriately, especially the attributes related spatial index creation.

Primary key for ArcSDE

ArcSDE requires every table registered has a unique index on a numeric column declared as type NUMBER(38).

Registration of spatial layers not created by ESRI client tools

The following are point, line and polygon examples that register spatial layers not created by ESRI client tools with ArcSDE data dictionary tables.

- Point example:

```
sdelayer -o register -l bts,shape -e p -C  
oid_,USER -i esri_sde -u oracle_username -p  
oracle_password
```

- Line example

```
sdelayer -o register -l road_geo,shape -e sl+ -  
C objectid,USER -i esri_sde -u oracle_username -  
p oracle_password
```

- Polygon example

```
sdelayer -o register -l buildings,shape -e a+ -  
C objectid,USER -i esri_sde -u oracle_username -  
p oracle_password
```

Deregistering layers from ArcSDE

If an ESRI client tool created the layer, then `sdelayer -o delete` will deregister and drop the table.

If an ESRI client tool did not create the table, then `sdelayer -o delete` will deregister the layer and **remove the entry in USER_SDO_GEOM_METADATA**, but will not drop the table. **If you want to re-register the spatial layer, you must repopulate USER_SDO_GEOM_METADATA.**

Here is an example that deregisters the building layer:

```
sdelayer -o delete -l buildings,shape -i  
esri_sde -u oracle_username -p oracle_password
```

Performance with ESRI client tools:

If you suspect a performance issue with an ESRI client tool first see the best practices for index creation. Another technique to consider is to try rendering with other visualization tools to see if the behavior is consistent across toolsets. If there is a performance discrepancy between the tools, review the best practices in this document to ensure your ESRI/SDO_GEOMETRY environment is optimal.

Oracle Database 11g significantly enhanced editing performance on SDO_GEOMETRY columns that have a spatial index. This may be another area worth testing.



Oracle Locator and Oracle Spatial 11g Best Practices
August 2008

Oracle USA
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2008, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

Other names may be trademarks of their respective owners.