Oracle Spatial 8.1.6 Performance-Related Characteristics

An Oracle Technical White Paper
May 2000

Contents

INTRODUCTION **1**

SUMMARY **1**

TEST METHODOLOGY **2**

ESTIMATING SPATIAL DATABASE SIZE **4**

TABLE SIZE 4
For ABI Data *5*
For Block Groups Data *5*
For Streams Data *9*
For Sample Test Data *10*
INDEX SIZE 11
SIZE COMPARISON BETWEEN ORACLE SPATIAL AND ESRI SHAPEFILES 12

LOAD PERFORMANCE **13**

LOADING POINT DATA 15
LOADING NON-POINT DATA 15
PARALLELISM AND PARTITIONING 16

INDEX BUILD CHARACTERISTICS **17**

UNDERSTANDING SPATIAL INDEXES 17
CHOOSING SPATIAL INDEX VALUES 19
Spatial Index Advisor *20*
CREATE INDEX STATEMENT 20
INDEX BUILD TIMES 21
SORT_AREA_SIZE PARAMETER CONSIDERATIONS 26
ANALYZE INDEX TABLE PERFORMANCE 28
SPATIAL INDEX SIZE 28

QUERY PERFORMANCE **31**

SDO_FILTER AND SDO_RELATE PERFORMANCE 31
ABI Data Set *32*
Block Groups Data Set *34*
Block Groups Filter Times *36*
Block Groups Relate Times *36*
Streams Data *38*

APPENDIX A - QUERIES TO DETERMINE SPACE REQUIREMENTS OF ORACLE SPATIAL **40**

ORACLE SPATIAL TABLES 40
ORACLE SPATIAL INDEXES 41

APPENDIX B - EXAMPLE CONTROL FILES **43**

APPENDIX C - USING SPATIAL INDEX ADVISOR **44**

APPENDIX D - SETUP FOR RUNNING SPATIAL INDEX ADVISOR..... 49

APPENDIX E - EXAMPLE QUERIES 51

ABI QUERY 51
BLOCK GROUPS QUERY 52
STREAMS QUERY 53

APPENDIX F - PERFORMANCE CHECKLIST AND HINTS 53

INTRODUCTION

This paper provides information to help you determine the characteristics of Oracle Spatial 8.1.6 in the context of a spatial data management solution. Information presented includes database size requirements, load times using SQL*Loader, index build times, query times, tuning suggestions, and a size comparison between Oracle Spatial data and ESRI shapefiles.

SUMMARY

Various operations were tested and analyzed using the object-relational model of Oracle Spatial release 8.1.6. The operations included loading spatial data, indexing spatial data, and using the spatial operators SDO_FILTER and SDO_RELATE.

The amount of *space required to load* spatial data can be estimated from load files. The amount of *space required to index* spatial data is highly variable and depends on several factors, including the type of spatial index (fixed or hybrid), the spatial index parameters (SDO_LEVEL and SDO_NUMTILES), the type of geometries (point or non-point), the length and/or area of the spatial data, and the position of the spatial data within the tiles generated for the index.

A sizing comparison is included that shows the difference in size between Oracle Spatial and ESRI shapefiles for the Block Groups data set. For the Block Groups data set, Oracle Spatial requires about 1.35 times as much storage as ESRI shapefiles.

The *speed of loading* Spatial Data using SQL*Loader depends on the type of data loaded (point data using SDO_POINT_TYPE, or other geometry types using SDO_ELEM_INFO_ARRAY and SDO_ORDINATE_ARRAY), as well as the average number of ordinates loaded per geometry. If speed is considered in rows loaded per second, point data loads faster than non-point data. However, if speed is considered in size of raw data loaded per second, very large geometries load faster.

The *speed of indexing* spatial data is related to the size and complexity of the geometries being indexed. Indexing (on a per-row basis) is fastest for point data. For non-point data, increasing the SDO_LEVEL and/or SDO_NUMTILES values increases the amount of time to build the index, as well as the storage requirements of the index. Increasing the SORT_AREA_SIZE parameter decreases the amount of time it takes to build an index. Note that with Oracle Spatial release 8.1.6, for optimal performance you must compute or estimate statistics on the spatial index table (guidance on how to do this is provided later in this document). For the data sets tested, the spatial index build is mostly a CPU intensive operation, with I/O to disk dominating at the beginning (when reading the spatial table) and at the end of the index build, when the spatial index is written to disk.

Timing information is presented for SDO_FILTER and SDO_RELATE operations. Some comparative data is presented based on varying the SDO_LEVEL and SDO_NUMTILES values. SDO_FILTER operations require less system resources than SDO_RELATE operations. SDO_RELATE operations usually consume disk resources at the beginning of queries, and then require significant CPU resources to perform the geometric operations associated with the relate operation.

TEST METHODOLOGY

All tests were run in the Spatial Laboratory at Oracle's New England Development Center.

All tests were run on a Sun E4000 system with a single 400Mhz CPU and 1.25 gigabytes of memory. Parallelization was not considered for these tests. The Oracle database SGA used only about 73 megabytes of memory on the system used for testing. SORT_AREA_SIZE was varied from 10 MB to 50 MB to determine the effect on performance of altering this parameter.

The majority of the test runs were completed using three different data sets:

- The American Business Information (ABI) data set contains point data representing business locations in the United States (with 10,279,241 points in the data set).

- The US Census Block Groups (BG) data set consists of 230,164 polygons.
- The California Streams (Streams) data set consists of 201,559 line strings.

Additional data sets have been used where noted.

All load times were determined using the SQL*Loader utility.

Each data set was tested using the object-relational model of Oracle Spatial with both hybrid and fixed indexing methods. The index build times were noted.

The SDO_LEVEL and SDO_NUMTILES index-creation parameters were varied for several tests.

For the ABI , Block Groups, and California Streams data sets, two Oracle Spatial operations were performed: SDO_FILTER to determine primary filter performance (which is an adequate measure of performance for some applications) and SDO_RELATE (mask=anyinteract) to return only the exact matching data set. These operations were chosen as representative of work done in a typical user environment.

For ABI and Block Groups tests, 9 windows were defined for the tests. The windows were all 32-vertex polygons that buffered a point in mid-town Manhattan with the following distances: 0.25, 0.50, 1.0, 2.0, 5.0, 10.0, 30.0, 50.0, and 100.0 miles. The following sequence of events was performed for each of the 9 windows:

1 Shut down and start the database to clear all caches and start all tests from a known starting point.

2 Run a window query two times for SDO_FILTER.

3 Shut down and start the database.

4 Run a window query two times for SDO_RELATE.

All times reported for ABI and Block Groups tests are for the second of the two window query runs.

Additional tests were done using the California Streams data set (both SDO_FILTER and SDO_RELATE operations).

ESTIMATING SPATIAL DATABASE SIZE

This section discusses considerations that affect table size and index size. It also discusses Oracle Spatial data and ESRI shapefiles in terms of size.

TABLE SIZE

It is possible to estimate the table size given some knowledge of the raw data to be loaded. For a table that has only a numeric primary key (GID) and the SDO_GEOMETRY object, the estimate can be done as follows:

GID	3 bytes plus 1 byte for every two numeric places
SDO_GTYPE	3 bytes plus 1 byte for every two numeric places
SDO_SRID	1 byte if NULL; or 3 bytes plus 1 byte for every two numeric places
SDO_POINT	1 byte if NULL; or for each of the three numeric values, 1 byte if NULL, or 3 bytes plus 1 byte for every two numeric places
SDO_ELEM_INFO	1 byte if NULL; or for each numeric value, 3 bytes plus 1 byte for every two numeric places, plus 40 bytes overhead for the VARRAY.
SDO_ORDINATES	1 byte if NULL; or for each numeric value, 3 bytes plus 1 byte for every two numeric places, plus 40 bytes overhead for the VARRAY.

Calculations based on this algorithm result in an estimated database size, assuming that all data is packed perfectly in the database with no overhead associated with the table or imperfect storage. To account for possible overhead, add a scaling factor of about 10% to the calculated size. This gives an approximation of the storage required for each table, except for the Block Groups table. The Block Groups table contains many very large

geometries, the largest of which (greater than 4KB) are stored in a separate LOB (large object) segment, and results in greater overhead for every row stored in the LOB segment.

For ABI Data

The following example shows data as it appears in the ABI point data file, which loads the GID and SDO_POINT columns:

```
1000000| 1| -86.032853| 39.920673|
```

To estimate the database size associated with the ABI data, the following calculation was used:

GID	= 3bytes + 4 bytes = 7 bytes
SDO_GTYPE	= 3 bytes + 1 byte = 4 bytes
SDO_SRID	= 1 byte (set to NULL)
SDO_POINT	= (3 + 4) + (3 + 4) + 1 = 15 bytes
SDO_ELEM_INFO	= 1 byte (set to NULL)
SDO_ORDINATES	= 1 byte (set to NULL)
Total	**= 29 bytes**

If you multiply 29 by the number of rows to be loaded (10,279,241), the resulting database size is approximately 284 MB. If you add a 10 percent scaling factor, the database size becomes 312 MB. (The actual database size in the tests was 298 MB.)

For Block Groups Data

The following example shows data as it appears in the Block Groups data set:

```
10000,3,,pt,,,,1,3,1;-86.47625,32.46743,-86.47691,32.46864,-
86.47761,32.47007,-86.478439,32.471649,-86.479294,32.472855,-
86.4799,32.4736, (…more data …):
```

To estimate the database size associated with the Block Groups data, the following calculation was used:

GID	= 3bytes + 3 bytes = 6 bytes
SDO_GTYPE	= 3 bytes + 1 byte = 4 bytes
SDO_SRID	= 1 byte (set to NULL)
SDO_POINT	= 1 byte (set to NULL)
SDO_ELEM_INFO	= 4 bytes + 4 bytes + 4 bytes + 40 bytes = 52 bytes
SDO_ORDINATES	= (avg. number ordinates * (3 + 4 bytes)) + 40 bytes = (227 * 7) + 40 bytes = 1629 bytes
Total	**= 1693 bytes**

If you multiply 1693 by the number of rows to be loaded (230,164), the resulting database size is approximately 371 MB. If you add a 10 percent scaling factor, the database size becomes 408 MB. However, the actual database size for the Block Groups table was higher (447 MB). This is due to LOB segments needed when the SDO_ORDINATES field on an SDO_GEOMETRY object exceeds 4000 bytes.

When creating tables that contain SDO_GEOMETRY columns with SDO_ORDINATES fields that may exceed 4000 bytes, it is important not to use the estimated database size calculated above (408 MB) as an initial extent of your table. This would waste a large amount of space because the initial extent allocation factors in the size of the LOB segment, but the LOB segment is actually stored outside the initial extent.

To optimize the storage clause, you can either guess at the size of the LOB segment or follow the procedure used for these tests: load the Block Groups table to determine the

size of the LOB segment compared to the size of the table data, then drop the Block Groups table and re-create it with more appropriate storage clauses. Detailed instructions for this procedure are in the following paragraphs.

After creating the Block Groups table (US_BG) initially, obtain the sizes of the base table and the LOB segments. To find all the space in megabytes (bytes/1024/1024) associated with the table and the LOB, the following statements were used:

```
-- SQL for space in megabytes of the US_BG table
SELECT bytes/1024/1024, segment_name, extent_id
   FROM user_extents
   WHERE segment_name = 'US_BG'
   ORDER BY segment_name, extent_id;
```

For the Block Groups table, this resulted in a table size of 207 MB.

Calculate the amount of space stored in the LOB segments. First, execute the following statement:

```
SELECT segment_name FROM user_lobs WHERE table_name='US_BG';
```

This statement returns two rows for each geometry type stored in the table: one row for the SDO_ELEMENT_INFO segment and one row for the SDO_ORDINATES segment. Then, for each of the segments returned by the preceding query, enter a statement such as the following:

```
-- SQL for space in megabytes for LOB segments
SELECT bytes/1024/1024,extent_id
   FROM user_extents
   WHERE segment_name = 'SYS_LOB0000021249C00009$$'
   ORDER BY extent_id;
```

For the SDO_GEOMETRY column in the Block Groups table, the size of the LOB segment was 240 MB and the size of the data segment was 207 MB, for a total table size of 447 MB.

You now have the information needed to create the table with optimal storage parameters. This can be done in one of two ways (with the US_BG table as the example):

- Specify a VARRAY storage clause for the LOB segment. For example:

```
CREATE TABLE US_BG (gid number, geometry mdsys.sdo_geometry)
       storage (initial 207M next 10M pctincrease 0)
       VARRAY geometry.sdo_ordinates store as lob
       (storage (initial 240M next 10M pctincrease 0));
```

- Create a table without a VARRAY storage clause, which allows the LOB segment to take the storage attributes of the user's default tablespace. The following example shows the creation of a tablespace, altering a user to use that default tablespace, then the table create command with a storage clause on the table, but the LOB objects use the storage attributes of the tablespace:

```
CREATE TABLESPACE US_BG_TBLSPC datafile
         '/usr/dev/oracle/us_bg.dbf' size 1500M
         default storage (initial 240M next 10M pctincrease
0);

 ALTER USER ORACLEUSER DEFAULT TABLESPACE US_BG_TBLSPC;

CREATE TABLE US_BG (gid number, geometry mdsys.sdo_geometry)
          tablespace US_BG_TBLSPC
          storage (initial 207M next 10M pctincrease 0);
```

For Streams Data

Estimating the data size for the Streams table, where each row in the data file is similar to the following:

```
201656,2,,pt,,,,1,2,1;-1321978,-1272456,-1321983,-1272437,-
1321944,-1272289,-1321934,-1272224,-1321955,-1272141,-
1321985,-1272097,-1322106,-1271979,-1322118,-1271918,-
1322107,-1271830,-1322139,-1271642,-1322196,-1271506,-
1322207,-1271357,-1322204,-1271245,-1322213,-1271168,-
1322209,-1271138:
```

To estimate the database size associated with the Streams data, the following calculation was used:

GID	= 3bytes + 3 bytes = 6 bytes
SDO_GTYPE	= 3 bytes + 1 byte = 4 bytes
SDO_SRID	= 1 byte (set to NULL)
SDO_POINT	= 1 byte (set to NULL)
SDO_ELEM_INFO	= (3 * (3 bytes + 1 bytes)) + 40 bytes = 52 bytes
SDO_ORDINATES	= ((avg. num. ordinates) * (3 + 4 bytes)) + 40 bytes = (38.9 * 7) + 40 bytes = 312 bytes
Total	**= 376 bytes**

If you multiply 376 by the number of rows to be loaded (201,659), the resulting database size is approximately 72 MB. If you add a 10 percent scaling factor, the database size becomes 79 MB. (The actual database size in the tests was 73 MB.)

The Streams data did not have SDO_ORDINATES fields larger than 4 KB. If it did have SDO_ORDINATES fields larger than 4 KB, the sizing analysis procedure would be similar to that for the Block Groups data.

For Sample Test Data

The following example shows data as it appears in a sample polygon test data set that was used:

```
26708413,3,,pt,,,,1,3,1;352026.3332381452,7449298.3493204669,3
61526.5219896557,5779197.7085787868,361523.8510542086,5779697.
6206780355,362023.7723017103,5779700.2913530963,352026.3332381
452,7449298.3493204669:
```

To estimate the database size associated with the sample test data, the following calculation was used:

GID	= 3bytes + 4 bytes = 7 bytes
SDO_GTYPE	= 3 bytes + 1 byte = 4 bytes
SDO_SRID	= 1 byte (set to NULL)
SDO_POINT	= 1 byte (set to NULL)
SDO_ELEM_INFO	= 3 * (3 bytes + 1 bytes) + 40bytes = 52 bytes
SDO_ORDINATES	= ((avg. num. vertices) * ((3 + 8 bytes)+(3 + 9 bytes))) +40 = (5 * 23 bytes) + 40 = 155 bytes
Total	**= 220 bytes**

If you multiply 220 by the number of rows to be loaded (191,453), the resulting database size is approximately 40 MB. If you add a 10 percent scaling factor, the database size becomes 44 MB. (The actual database size in the tests was 40 MB.)

The sample test data did not have SDO_ORDINATES fields larger than 4 KB. If it did have SDO_ORDINATES fields larger than 4 KB, the sizing analysis procedure would be similar to that for the Block groups data.

INDEX SIZE

Many factors affect the size of Oracle Spatial indexes. For non-point data, higher values for the indexing parameters SDO_LEVEL and SDO_NUMTILES increase the amount of storage required. The shape, area, length, and positioning of the geometries within the generated tiles affect the spatial index size.

The spatial index consists of two pieces:

1) A spatial index table generated by the CREATE INDEX statement. (This table is referred to as *Index Table* in the following tables in this section.)

2) Two B-tree indexes to provide rapid access to the information in the table described in 1). (These indexes are referred to as *B-tree1* and *B-tree2* in the following tables in this section.)

The following tables show the sizes (in megabytes) of the index components for the ABI, Block Groups, and Streams data sets using the specified SDO_LEVEL and SDO_NUMTILES values.

ABI Data Set (MB)

	SDO_LEVEL=13	SDO_LEVEL=18	SDO_LEVEL=13, SDO_NUMTILES=1	SDO_LEVEL=18, SDO_NUMTILES=1
Index Table	253	265	376	386
B-tree1	325	336	398	428
B-tree2	248	248	248	248
Total	**826**	**849**	**1032**	**1062**

Block Groups Data Set (MB)

	SDO_LEVEL=13	SDO_LEVEL=14	SDO_LEVEL=13, SDO_NUMTILES=8	SDO_LEVEL=14, SDO_NUMTILES=8
Index Table	53	160	100	227
B-tree1	68	209	122	279
B-tree2	52	157	77	235
Total	**173**	**526**	**299**	**741**

Streams Data Set (MB)

	SDO_LEVEL,SDO_NUMTILES							
	8,0	8,6	9,0	9,6	10,0	10,6	11,0	11,6
Index Table	8	35	11	36	16	38	28	46
B-tree1	10	43	14	44	21	47	36	56
B-tree2	8	30	11	29	17	31	28	37
Total	**26**	**108**	**36**	**109**	**54**	**116**	**92**	**139**

SIZE COMPARISON BETWEEN ORACLE SPATIAL AND ESRI SHAPEFILES

An ESRI Shapefile consists of three files: a shape file, which holds the spatial data; an index file, which provides access to the spatial data; and an attribute file, which holds the information associated with the spatial data.

The Block Groups data consisted of 3324 ESRI Shapefiles, with the number and total sizes of each type of file shown in the following table.

Data file type	Number of files	Size in MB
Shape	3234	442
Index	3234	5
Attribute	3234	11
Total	**9702**	**458**

In Oracle Spatial 8.1.6, the size of the Block Groups spatial data and attributes is 447 MB and the index size at SDO_LEVEL 13 is 173 MB, for a total size of 620 MB. This is about 1.35 times the size of the ESRI Shapefiles. The difference, however, may not apply to other scenarios due to the large number of VARRAY columns stored in LOB segments for the Block Groups data. Other Oracle Spatial tables may be closer in size to ESRI Shapefiles. It is also important to note that ESRI shapefile users may have to deal with many files (over 9000 files for Block Groups data), as opposed to a single seamless table with Oracle Spatial. The ESRI Shapefile is a very common format and is simply used here to illustrate storage requirements.

LOAD PERFORMANCE

SQL*Loader performance varies with the number of vertices in the geometry. For each row, the geometry and a single numeric ID field were loaded.

All tables were created using storage parameters to define initial and next extents, as in the following example:

```
CREATE TABLE abi_8i (gid number, geometry mdsys.sdo_geometry)
  pctfree 0 storage (initial 297M next 10M pctincrease 0);
```

Note the following about this example:

- The parameter PCTFREE is used so that Oracle completely fills the pages on disk with data. If this parameter is not specified, then the Oracle default is to leave 10% of the

space empty in each page. Do not set PCTFREE to 0 if there will be many new records added to the table after the initial load.

- The storage parameter INITIAL is used to pre-allocate space within Oracle so that the table will not have to be extended during the load process. If there are many extents then query performance may also be affected.

- The parameter NEXT is the size to make any additional extents after the initial extent is filled with data. The initial and next extents should be sized according to the data loaded, and what the workload will look like. If large amounts of data are added constantly it would be better to create the table with larger initial and next extents.

- The parameter PCTINCREASE is set to 0 to prevent the extents from being created with every new extent being the size of the previous extent plus some percentage. This can cause the database to have large amounts of wasted space when the extents get very large.

- The use of the storage parameter MAXEXTENTS should also be considered for tables that will have large amounts of data added to them. If the parameter is not specified, the Oracle default of 121 is used.

Information about each of the data sets loaded is in the following tables:

Data Set Information

Data Set	Raw Data Size (bytes)	Average Number of Vertices/Geometry	Number of Rows Loaded	Loaded Data Size (MB)
ABI	424,727,597	1	10279241	298
TEST1	39,992,031	5	191453	44
STREAMS	71,782,031	19.45	201659	73
BG	538,102,288	113.56	230164	447

Load Rates

Data Set	Load Time (seconds)	Load Rate (rows/sec)	Load Rate (vertices/sec)	Load Rate (raw KB/sec)
ABI	2162	4755	4755	192
TEST1	155	1235	6175	252
STREAMS	267	755	14690	263
BG	1160	198	22532	453

LOADING POINT DATA

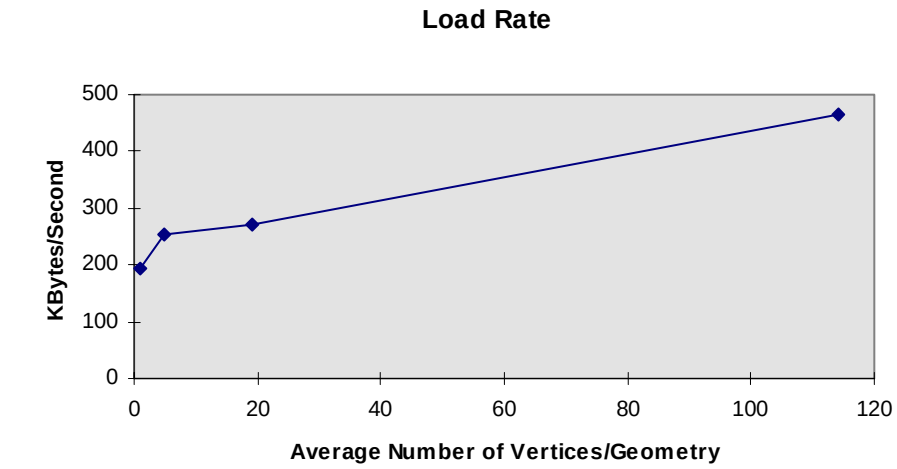
The ABI data set is the only data set tested that loads data into the SDO_POINT field of the SDO_GEOMETRY data type. On the tested system, approximately 4755 rows/second were loaded, equivalent to 195 KB/second from the raw data file. In terms of rows/second, the ABI data set performance is the best of all the load tests; however, in terms of raw data loaded/second, larger geometry objects provide better load performance.

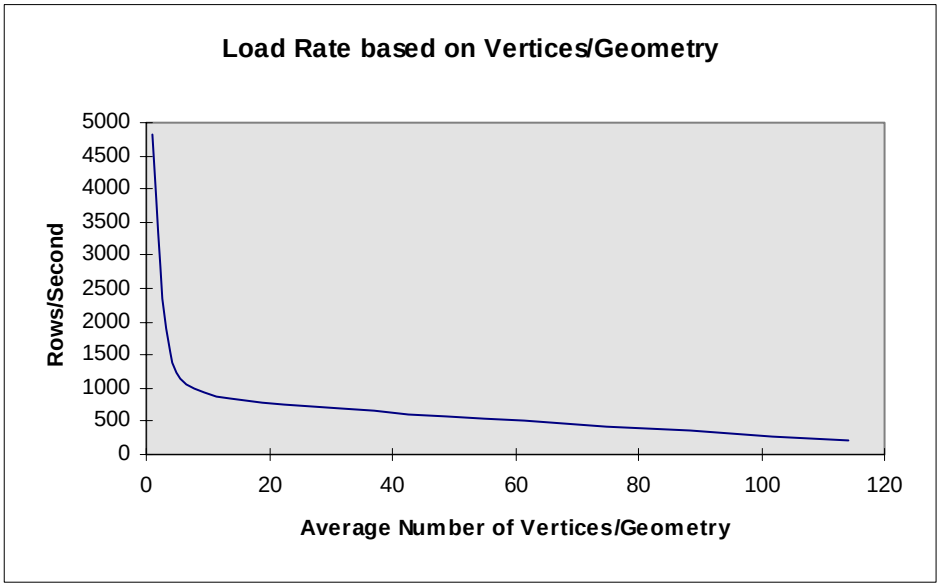
The ABI example in Appendix A shows the data as it appears in the ABI point data file, which loads the GID and SDO_POINT columns.

LOADING NON-POINT DATA

Appendix A shows an example control file for loading non-point data. When loading data using the SDO_ORDINATES field, the larger the average geometry size (more vertices), the faster the load rate in terms of raw Kb/sec, but the slower the load rate in terms of rows/second. These relationships are shown in the Load Rates Table (above) and in Figure 1.

Figure 1: Load Rate: KBytes/Second and Rows/Second





PARALLELISM AND PARTITIONING

There is currently no parallelism built into Oracle Spatial that allows it to decompose single queries or index builds to run on more than a single processor. This capability is

planned for a future release of Oracle Spatial. Multiple CPUs are used when multiple spatial queries are executing concurrently.

Oracle Spatial creates and uses indexes based on the Oracle8*i* extensible indexing feature. Currently, there is no mechanism available to propagate partitioning column information to extensible index tables. This capability is planned for a future release of Oracle Spatial.

INDEX BUILD CHARACTERISTICS

Spatial indexes were built on the ABI, Block Groups, and Streams data sets.

Query performance is highly dependent on a number of factors, including the values used for SDO_LEVEL (for fixed indexes, recommended in almost all cases), SDO_LEVEL and SDO_NUMTILES (for hybrid indexes), and the size and complexity of the geometries. Because the size and complexity of the spatial data are usually predetermined, the most important part of building a spatial index is related to choosing the best value for SDO_LEVEL. For non-point data, the SDO_LEVEL value directly affects the storage requirements for the table and build time for the spatial index.

Oracle Enterprise Manager has been enhanced to include the Spatial Index Advisor, which helps in the spatial index creation and analysis process.

UNDERSTANDING SPATIAL INDEXES

Before discussing the details of choosing the correct tuning parameters, it is worthwhile to briefly discuss what a spatial index is, and how choosing the correct values when creating the index affects query performance

A spatial index is a specific type of index created by Oracle Spatial with information about where geometries are in a coordinate space. The coordinate space is divided into rectangles called *tiles* when an index is built, and the spatial index contains one entry for each tile a geometry touches. If a geometry is a point, for example, it will only touch one tile so there will only be one entry in the spatial index for each point. If the data is a long

line string and touches many tiles, there will be many index entries (one for each tile) for that geometry. The size of the tiles is based on how many tiles are created for the coordinate space, which is determined by the SDO_LEVEL parameter value when the spatial index is created. When SDO_LEVEL is the only tiling information specified when an index is created, the index is a *fixed* index. (Not discussed here is the *hybrid* index, where the SDO_NUMTILES parameter is specified to give additional tiling information when the index is created.)

The spatial index is used to approximate spatial query results by determining if the tiles associated with a query geometry (usually an area of interest) interact with any of the tiles associated with the geometries in a layer. This approximation is a relatively "inexpensive" computational operation, and is called a *primary filter* (or, SDO_FILTER) operation. However, when exact (not merely approximate) answers are needed for a spatial query, the computationally "expensive" geometric calculations are made based on the approximate results returned by the primary filter. The query that returns the exact answer is known as a *secondary filter* (or, SDO_RELATE) operation. (For a detailed discussion of spatial indexes, see the *Oracle Spatial User's Guide and Reference*).

A new optimization in Oracle Spatial 8.1.6 enables the primary filter to pass some final results about certain geometries to the secondary filter. The secondary filter can automatically accept these results without doing geometric calculations. In Oracle Spatial 8.1.6 this optimization is only for the ANYINTERACT mask (the most common mask), but the optimization is planned for other masks in a future release. In this optimization, when an area of interest is tiled and some of the tiles touch only the interior of the geometry representing the area of interest, any geometry compared with that area of interest can automatically be marked to pass the secondary filter if any tile associated with that geometry touches an interior tile of the area of interest. This optimization can speed up secondary filter operations tremendously.

There are several considerations when creating a spatial index: The first is that if the tiles stored in the index are a good approximation for query geometries (areas of interest), the primary filter will return answers closer to the final results. Another consideration is the

trade-off between having the index geometries closely approximate query geometries on the one hand, and the increased resource requirements on the other hand: storage requirements for the index, index build time, and query time associated with the logical index. The higher the value for SDO_LEVEL, the tighter is the approximation of index geometries to query geometries, but also more index entries are stored for each geometry and performance can slow due to the larger spatial index. Finally, when an area-of-interest is indexed (the tiles are created), it is good to have interior tiles so that further interior tile optimizations can be accomplished. As a general rule, approximately 200 tiles for an area of interest is a good starting point. This number is sufficient for most areas of interest to ensure that there are interior tiles, and it allows for zooming in or out with reasonable results.

CHOOSING SPATIAL INDEX VALUES

Under most circumstances, Oracle recommends using fixed indexes (that is, specifying only SDO_LEVEL and not SDO_NUMTILES) when indexing spatial data. The rare circumstances where hybrid indexes should be considered for use are as follows:

1) When joins are required between layers whose optimal fixed index level (SDO_LEVEL) values are significantly different (4 levels or more), it may be possible to get better performance by bringing the layer with a higher optimal SDO_LEVEL down to the lower SDO_LEVEL and adding the SDO_NUMTILES parameter to ensure adequate tiling of the layer. The best starting value for SDO_NUMTILES in the new hybrid layer can be calculated by getting a count of the rows in the spatial index table and dividing this number by the number of rows with geometries in the layer, then rounding up. A spatial join ('QUERYTYPE=JOIN') is not a common requirement for applications, and it is comparable to a spatial cross product where each of the geometries in one layer will be compared to each of the geometries in the other layer.

2) When both of the following are true for a single layer:

- The layer has a mixture of many geometries covering a very small area and many polygons covering a very large area.
- The optimal fixed tiling level for the very small geometries will result in an extremely large number of tiles to be generated for the very large geometries, causing the spatial index to grow to an unreasonable size.

If both of these conditions are true, it may be better to use the SDO_NUMTILES parameter to get coverage for the smaller geometries, while keeping the fixed tile size relatively large for the large geometries by using a smaller SDO_LEVEL value.

Determining the best value for SDO_LEVEL is very important. If SDO_LEVEL is set too small, the tiles will be too large and an accurate approximation for each geometry will not be achieved, and the primary filter operation will have poor selectivity (potentially sending too many candidates to the secondary filter). If SDO_LEVEL is set to too high, the spatial index table may grow to an unreasonable size.

Spatial Index Advisor

Oracle recommends using Spatial Index Advisor to help with the analysis and creation of Oracle Spatial indexes. More information about using Spatial Index Advisor can be found in Appendix C.

CREATE INDEX STATEMENT

With Oracle Spatial 8.1.6, Oracle recommends that you specify the initial and next extents of both the spatial index table and the B-tree indexes on the spatial index table. As previously mentioned, analyzing the table for optimizer statistics is required to ensure good performance. For example:

```
CREATE INDEX us_bg_fxidx ON us_bg (geometry)
      INDEXTYPE IS MDSYS.SPATIAL_INDEX
      PARAMETERS ('SDO_LEVEL=13 INITIAL=50M NEXT=5M
```

```
        PCTINCREASE=0 BTREE_INITIAL=52M BTREE_NEXT=17M
        BTREE_PCTINCREASE=0 TABLESPACE=BENCH');
ANALYZE TABLE  us_bg_fxidx_fl13$
     ESTIMATE STATISTICS SAMPLE 1 PERCENT;
```

The preceding statements create a fixed spatial index at SDO_LEVEL 13, allocating an initial size of 50 MB to the table with each additional extent added to the table being 5 MB. Setting PCT_INCREASE to 0 ensures that each additional extent is 5 MB and that extent size does not grow. Also, the statement makes sure the B-tree indexes have a fixed initial size and extend size, and all storage is to a tablespace called BENCH.

INDEX BUILD TIMES

Building an Oracle Spatial index is a CPU-intensive operation. Most of the time spent building indexes is for the calculations to tile each geometry. At the end of the index build operation, there is significant I/O activity to disk as the index table is written and B-tree indexes are built on the Spatial index table.

The factors that affect the speed at which a spatial index for a layer is built which are user definable include the SDO_LEVEL value chosen for the layer, and also the SDO_NUMTILES value if the index is a hybrid index. Additionally, increasing the SORT_AREA_SIZE parameter value in the Oracle initialization file (init.ora) can improve performance during spatial index creation.

The comparisons in this section use the ABI and Block Groups data sets, which have the same spatial reference system and the coordinate system extents (longitude/latitude data with extents of -180 to 180 in the x coordinate and -90 to 90 in the y coordinate).

The effect on spatial index build performance of increasing the SDO_LEVEL value for point-only layers such as the ABI layer is very minimal. The effect of increasing SDO_LEVEL on spatial index build performance for a layer like Block Groups, with very complex geometries, is very pronounced, as shown in Figure 2.

Figure 2: Spatial Index Build Times with Changing SDO_LEVEL: ABI and BG

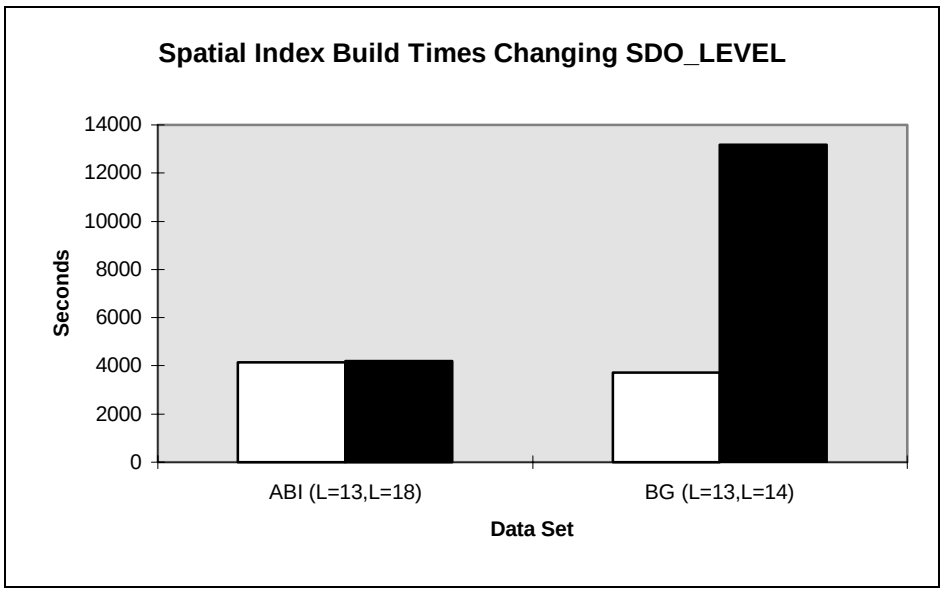
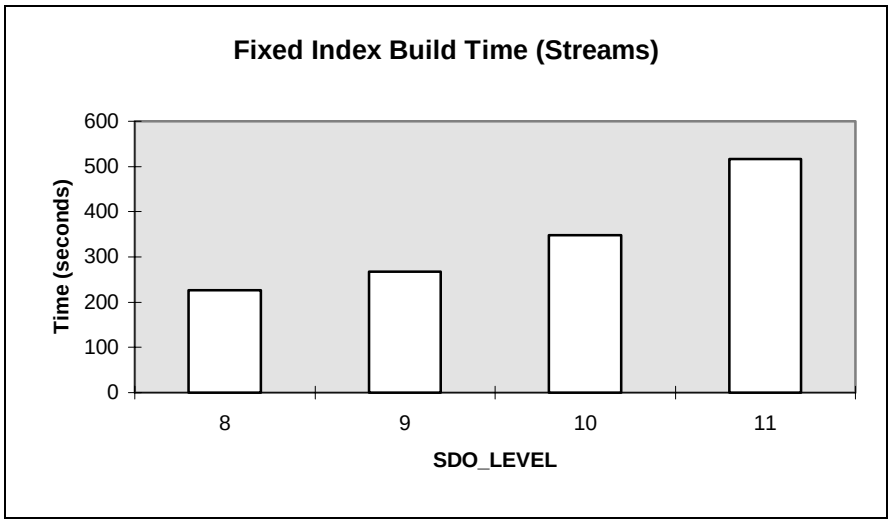


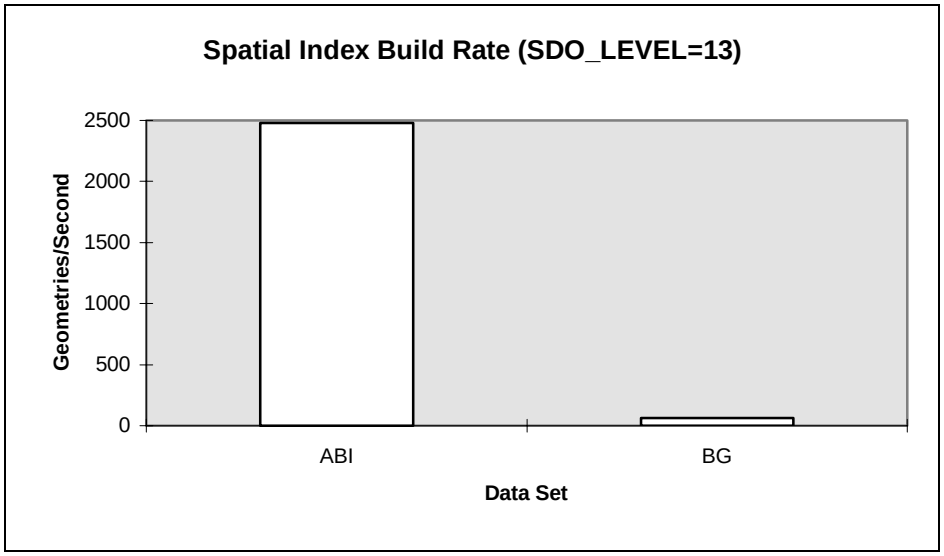
Figure 3 supplements Figure 2 by showing how spatial indexing time for the Streams data grows as SDO_LEVEL is increased from 8 to 11.

Figure 3: Spatial Index Build Times with Changing SDO_LEVEL: Streams



The complexity of the data in the layer affects the indexing rate. The indexing rate in rows per second is much higher for less complex data (such as the ABI point data), as shown in Figure 4.

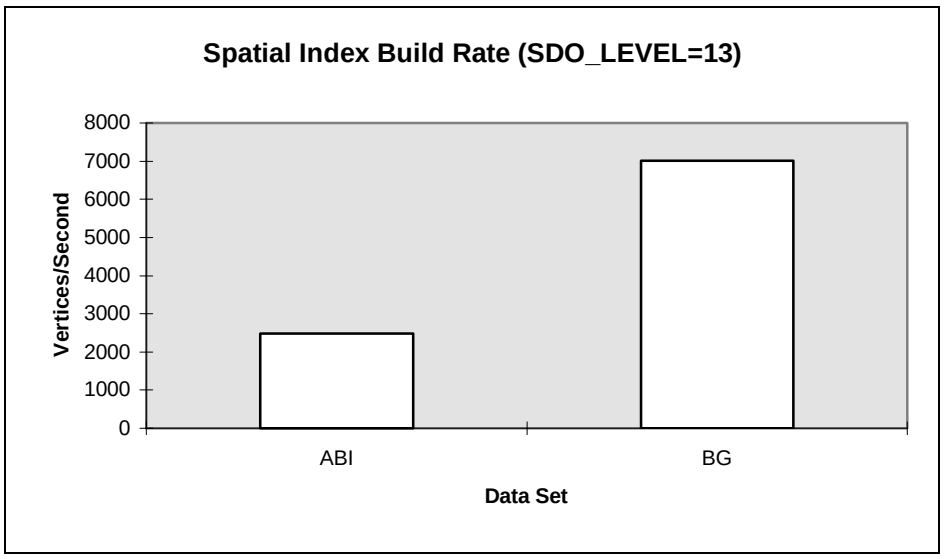
Figure 4: Index Build Rate and Geometry Complexity: Rows/Second



The Streams data was indexed at a rate of approximately 890 geometries/second, although no direct comparison can be made with ABI or BG data due to the different indexing level and spatial extents.

However, in vertices per second, the indexing rate is higher for more complex data (such as the Block Groups data), as shown in Figure 5.

Figure 5: Index Build Rate and Geometry Complexity: Vertices/Second

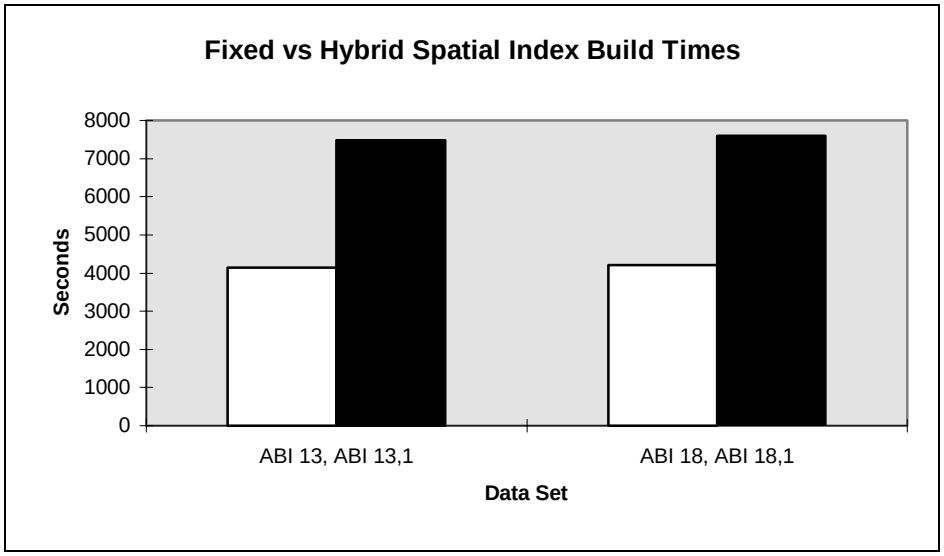


Looking at the vertices/second indexing rate, however, does not tell the whole story in terms of how long it will take to index a layer. Another part of determining indexing rate deals with the shapes and sizes of the geometries represented by the data. Taking a subset of the Block Groups data within a 100-mile radius of central Manhattan returns about 10% of the entire Block Groups data set. The average number of vertices/geometry is approximately 44. If vertices/geometry were the only consideration, the expected indexing rate for the layer would be somewhere between 2478 vertices/second (ABI, 1 vertex/geometry) and 7007 vertices/second (Block Groups, average 113 vertices/geometry). In fact, the subset of the Block Groups data was spatially indexed in about 32 seconds, giving a much higher spatial indexing rate of approximately 32478 vertices/second. This rate was achieved due to both the lack of complexity of the geometries and the smaller area taken up by each of the geometries. Geometries with larger areas require more CPU time for the spatial calculations for each tile, as well as more index space because every tile (interior and boundary) for every geometry is stored in the spatial index.

Spatial indexes require more time for building hybrid indexes than fixed indexes. For example, for ABI data, even though there is always just one spatial index entry for each

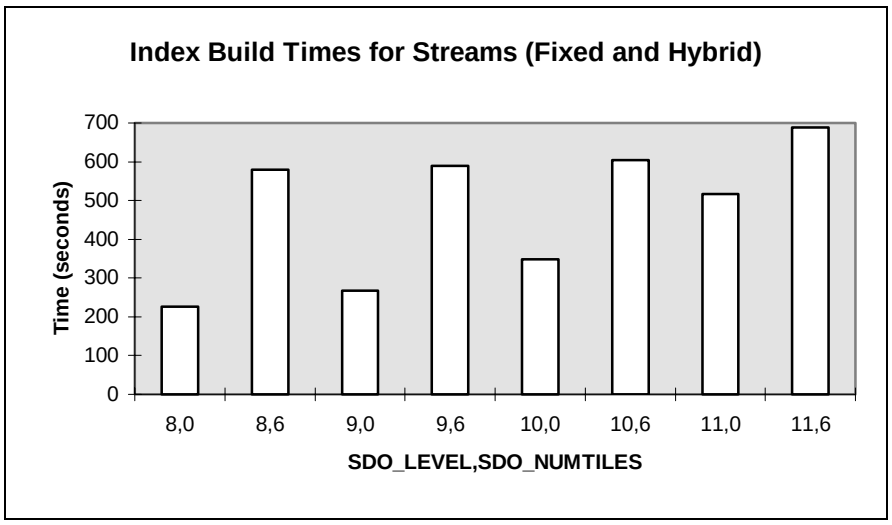
point regardless of whether the spatial index is fixed or hybrid (SDO_NUMTILES=1), it takes over 70% longer to build the hybrid index than the fixed index, as shown in Figure 6.

Figure 6: Fixed vs. Hybrid Index Build Times



Tests with the Streams data show how index build times change depending on the SDO_LEVEL value and whether the index is fixed or hybrid, as shown in Figure 7.

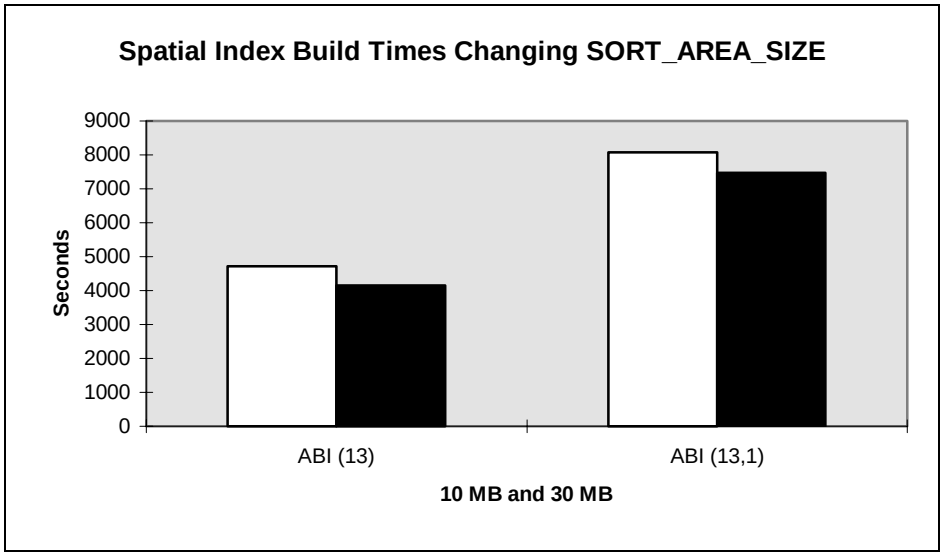
Figure 7: SDO_LEVEL and Fixed vs. Hybrid Index Build Times



SORT_AREA_SIZE PARAMETER CONSIDERATIONS

Every spatial index build includes the creation of B-tree indexes on the spatial index table. Increasing the SORT_AREA_SIZE database parameter affects the spatial index build time. For data sets of this size, with an SGA size of about 73 MB, setting the SORT_AREA_SIZE to 30 MB gives better index build performance than when the SORT_AREA_SIZE is 10 MB, as shown in Figure 8.

Figure 8: SORT_AREA_SIZE and Index Build Time



However, the performance benefit does not grow if SORT_AREA_SIZE is further increased without changes to any other SGA and memory-related parameters. For example, if no other parameters are changed in this setup, a SORT_AREA_SIZE value of 50 MB results in worse performance than a value of 30 MB, and can be worse than a value of 10 MB. Further investigation and experimentation would be needed to determine the optimal mix of parameter values in any given system environment.

When querying point data, SORT_AREA_SIZE should not be a consideration with respect to the spatial portion of the query. When the LAYER_GTYPE=POINT parameter is specified for a query, Oracle Spatial does not perform a DISTINCT operation when querying the index table (because there is exactly one entry in the spatial index table for each point). Because there is no DISTINCT operation in the spatial index table query, no sort operation is required -- as opposed to other (non-point) spatial data which could have more than one entry in the spatial index table (there is one entry for each tile that a geometry comes in contact with), and where the DISTINCT operation will cause a sort to occur.

The following table shows the index build times in seconds for SORT_AREA_SIZE values of 10 MB, 30 MB, and 50 MB with the ABI and Block Groups data and different fixed and variable index definitions:

- ABI, fixed index (SDO_LEVEL=13)
- ABI, fixed index (SDO_LEVEL=18)
- ABI, hybrid index (SDO_LEVEL=13, SDO_NUMTILES=1)
- ABI, hybrid index (SDO_LEVEL=18 SDO_NUMTILES=1))
- Block Groups, fixed index (SDO_LEVEL=13)
- Block Groups, fixed index (SDO_LEVEL=14)
- Block Groups, hybrid index (SDO_LEVEL=13, SDO_NUMTILES=8)
- Block Groups, hybrid index (SDO_LEVEL=14 SDO_NUMTILES=8))

Index Build Time (Seconds)

SORT_AREA_SIZE	ABI 13	ABI 18	ABI 13,1	ABI 18,1	BG 13	BG 14	BG 13,8	BG 14,8
10M	4708	4866	8083	8654	3856	13437	5140	14581
30M	4147	4206	7479	7672	3730	13165	4888	14482
50M	4652	4803	8651	8740	3769	13411	5011	14593

ANALYZE INDEX TABLE PERFORMANCE

For each of the data sets tested, the ANALYZE TABLE command required very little time to run, and in no case exceeded 1 minute.

SPATIAL INDEX SIZE

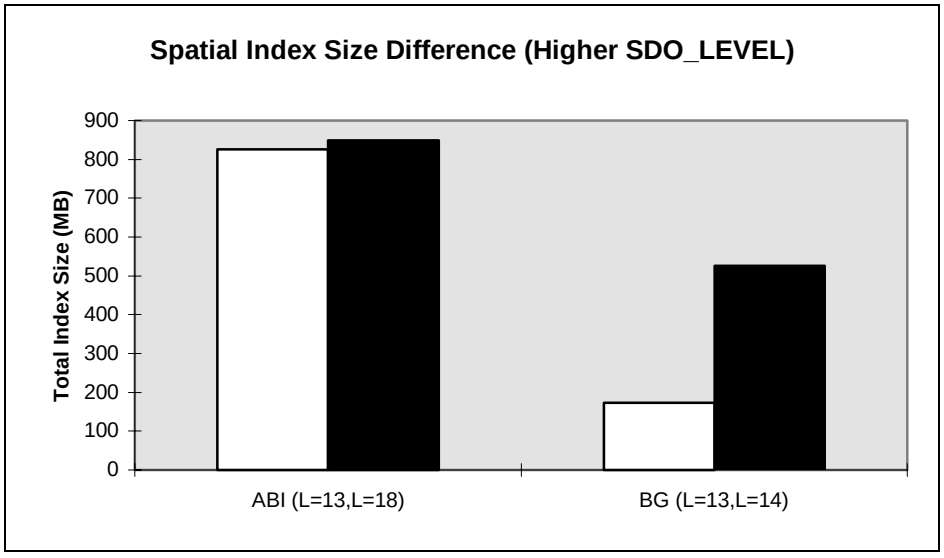
As SDO_LEVEL increases, the size of the created spatial index table and B-tree indexes grows. There are two reasons for this growth:

- The encoding scheme for the tiles causes the data in the code to increase in size. (For point data, this is the only factor associated with the increase in size.)

- For geometries that are not point-only, each tile associated with that geometry (including the area for geometries with area) must be stored in the spatial index. Large tiles (small SDO_LEVEL values) mean fewer entries in the index per non-point geometry. For smaller tiles (larger SDO_LEVEL values), more tiles are associated with each geometry, and the index size grows.

As Figure 9 shows, for point-only data (ABI) increasing the SDO_LEVEL value (even substantially, from 13 to 18) results in an almost negligible increase in the spatial index size; however, for complex geometries (Block Groups) increasing the SDO_LEVEL even slightly (from 13 to 14) results in a substantial increase in the spatial index size.

Figure 9: SDO_LEVEL and Index Size (Fixed Index)

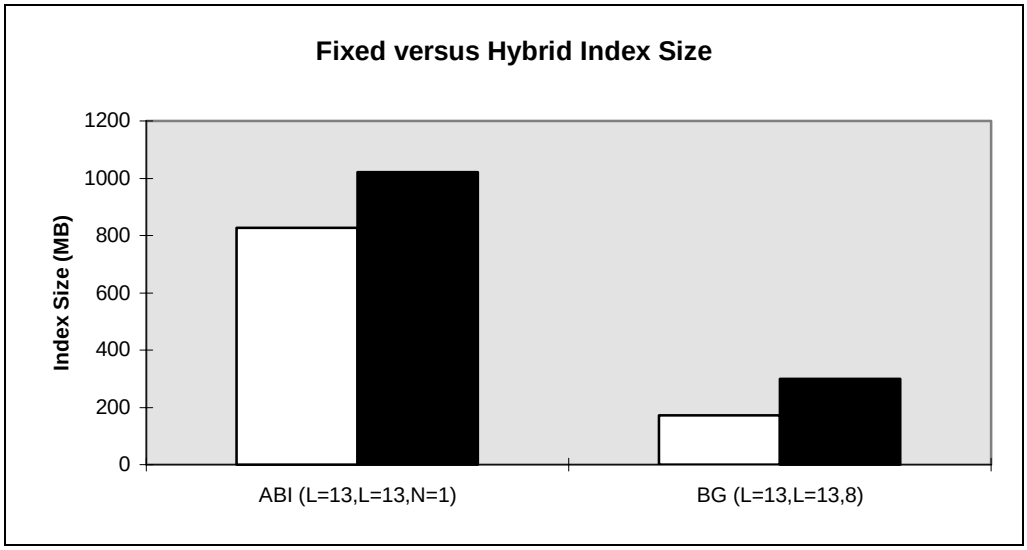


Hybrid spatial indexes are larger than fixed spatial indexes because a hybrid index table stores an additional column for each entry in the index. Also, a hybrid index table grows in size as the SDO_LEVEL level increases for the same reason that fixed spatial indexes grow. Because the SDO_NUMTILES value specifies a minimum number of spatial index table entries for each geometry, the index table can grow even larger.

As Figure 10 shows, a hybrid index is larger than a fixed index for both point-only data (ABI) and complex geometries (Block Groups). In all four tests the SDO_LEVEL value was

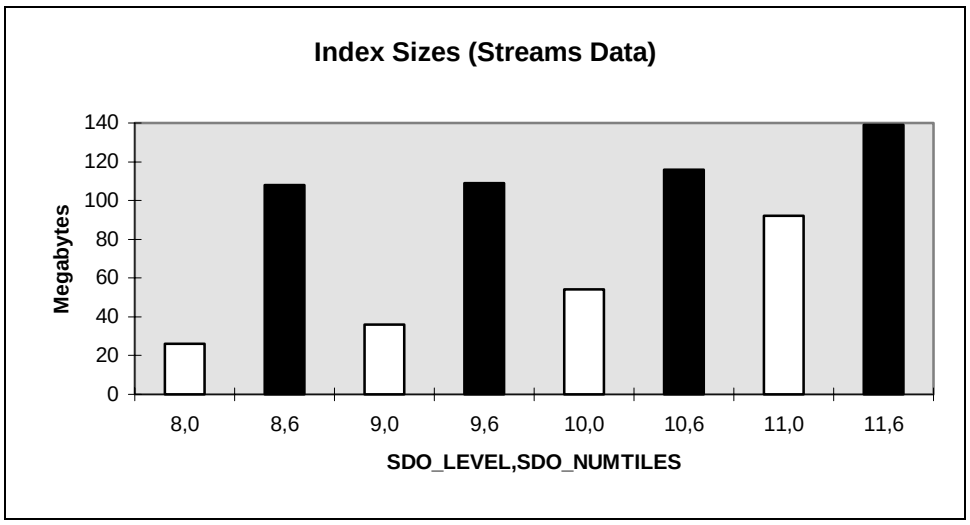
13. For the hybrid indexes, the SDO_NUMTILES value was 1 for the ABI data and 8 for the BG (Block Groups) data.

Figure 10: Index Size Increase for Fixed vs. Hybrid Spatial Index



With the Streams data, an increase in the SDO_LEVEL value and a change from fixed to hybrid indexing each caused an increase in the index size, as shown in Figure 11.

Figure 11: Index Size Increase for Fixed vs. Hybrid Spatial Indexes



QUERY PERFORMANCE

The performance of SDO_FILTER and SDO_RELATE queries was tested.

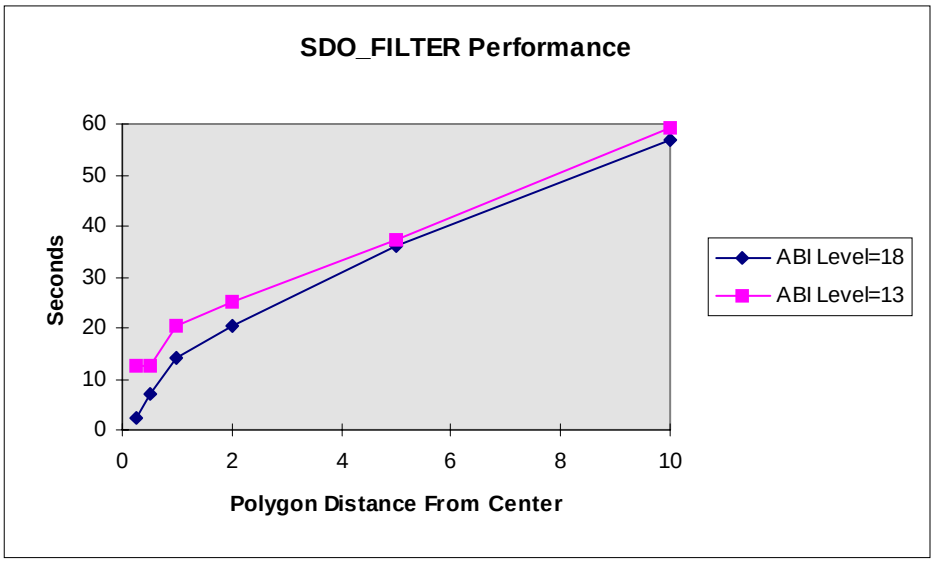
SDO_FILTER AND SDO_RELATE PERFORMANCE

SDO_FILTER and SDO_RELATE operations were run using the Block Groups and ABI data sets. The query window was always defined to be a 32-vertex polygon, with the window area set as the polygon with a distance of 0.25 miles to 100.0 miles from the center of Manhattan in New York City. SDO_FILTER operations require less system resources than SDO_RELATE operations, and tend to be more balanced in their disk and CPU requirements. SDO_RELATE operations usually consume disk resources (do lots of I/O) at the beginning of the queries while reading geometry information from disk, and then require significant CPU resources to perform the geometric operations associated with the relate operation.

ABI Data Set

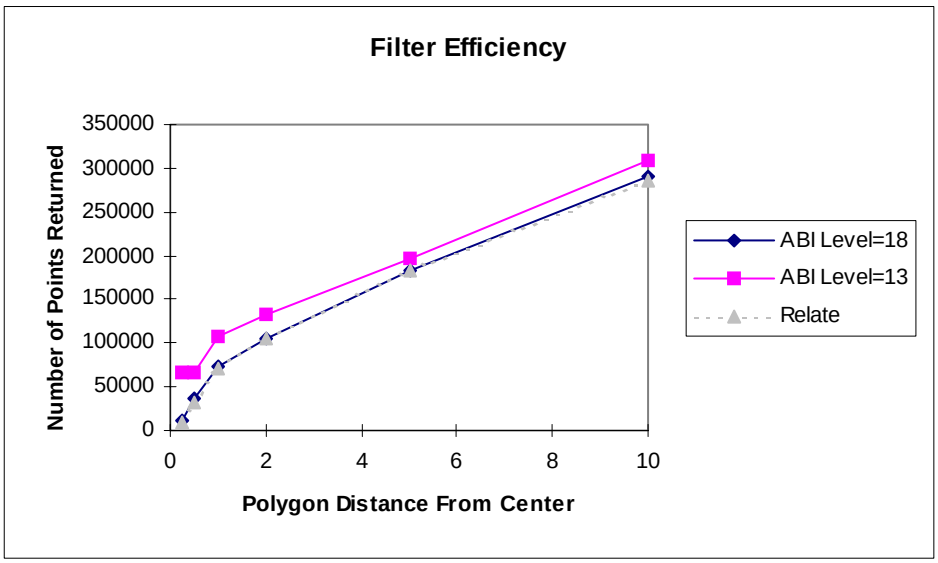
For the ABI point data, there is little or no difference in the performance between using fixed and hybrid indexes. For small distances (radius of 2 miles or less from the center of Manhattan), increasing the SDO_LEVEL from 13 to 18 caused a significant improvement in performance; however, for larger distances (radius of 10 miles or more from the center of Manhattan), this increase in the SDO_LEVEL value caused very little difference in performance. Figure 12 illustrates the performance improvement for distances up to 10 miles for the filter operation when index was created at SDO_LEVEL=18 compared to when the index was created at SDO_LEVEL=13.

Figure 12: SDO_FILTER Performance with Increased SDO_LEVEL (ABI Data)



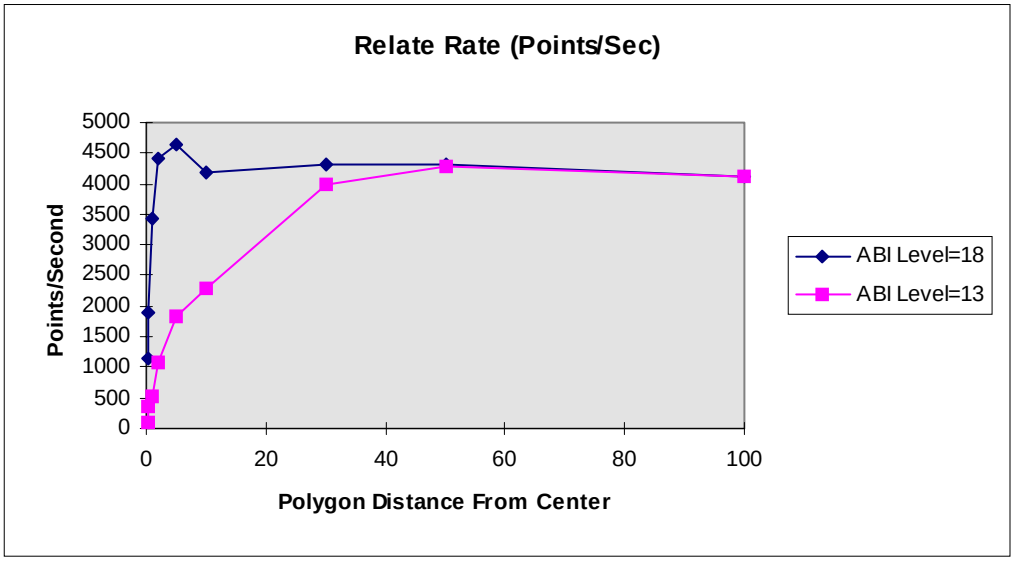
Note that not only was the time to complete the filter operation lower at SDO_LEVEL=18, but also the selectivity (efficiency) of the filter was much greater. As Figure 13 shows, the filter operation returns almost the same number of rows as the relate operation when the SDO_LEVEL value is 18.

Figure 13: SDO_FILTER Efficiency with Increased SDO_LEVEL (ABI Data)



The relate (SDO_RELATE) rate improves as the query windows grow larger, resulting in a larger number of interior tiles for the query window. This means that many points can be accepted by the SDO_RELATE operation based on the interior tile optimization discussed previously. As Figure 14 shows, the relate rate with the ABI data is significantly better for SDO_LEVEL values of 18 as opposed to 13 for distances up to around 20-30 miles from the center of Manhattan.

Figure 14: SDO_LEVEL Effect on SDO_RELATE Performance (ABI Data)



Block Groups Data Set

Using the methodology explained in the "Choosing Spatial Index Values" section (page 19), the right fixed index choice for the Block Groups data set was between SDO_LEVEL 13 and 14. Looking at the data in the Spatial Index Advisor shows that at SDO_LEVEL 14, the data is optimized for window queries where the window is a circle with a radius between 30 and 50 miles whose center is in central Manhattan.

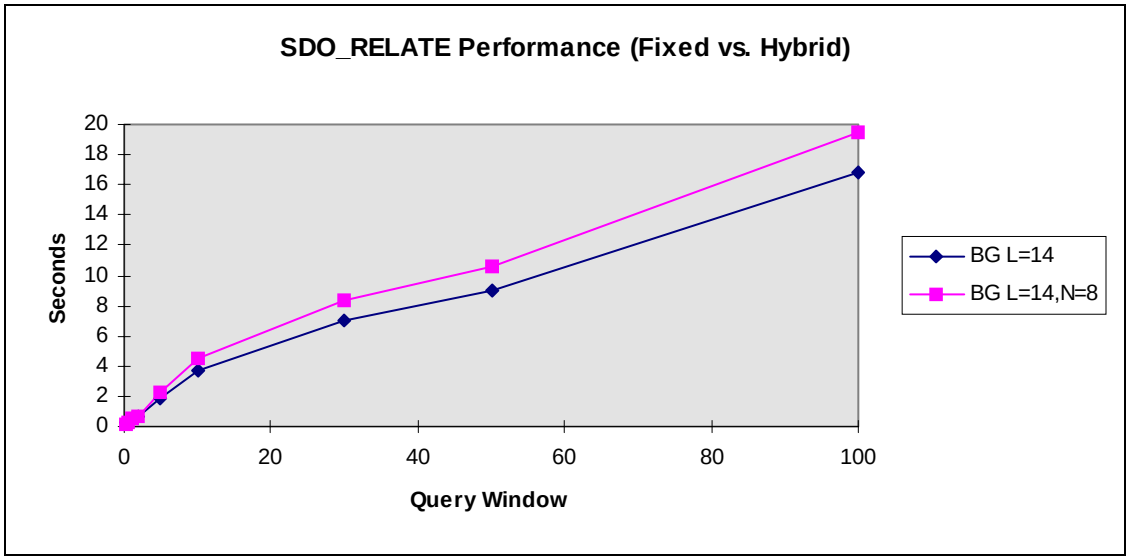
As shown in the sections on index creation and sizing, index creation time and index size increase greatly as the SDO_LEVEL value increases from 13 to 14. Increasing the SDO_LEVEL from 13 to 14 caused the index size to increase over 3 times and the index build time to increase over 4 times.

For query performance, though, for most of the SDO_FILTER and SDO_RELATE queries the performance was slightly better when the tests were run with indexes built at SDO_LEVEL=14 rather than SDO_LEVEL=13.

When hybrid indexes were tested with Block Groups data, for the smallest query windows the hybrid index built with SDO_LEVEL=14 and SDO_NUMTILES=8 provided

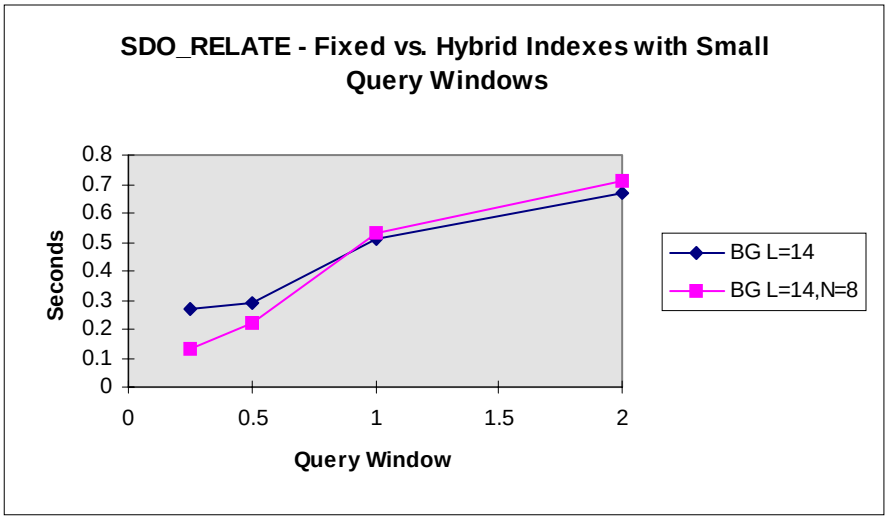
the best performance. However, performance degraded for hybrid indexes compared to fixed indexes as the query window increased, as shown in Figure 15.

Figure 15: SDO_RELATE Performance with Hybrid Indexes (BG Data)



Only with very small query windows do hybrid indexes provide any query performance advantage. Figure 16 focuses on the results from Figure 15, but just for the 0.25 to 2 mile radius queries.

Figure 16: Query Window Size and Performance with Hybrid Indexes (BG Data)



Block Groups Filter Times

The following table shows the filter times (in seconds) for all of the Block Groups queries tested. Note that hybrid indexes provide for better selectivity only when the query windows are very small, and that the extra rows in the hybrid tables degrade query performance as the query window gets larger.

Block Groups Filter Times (Seconds)

Query window	SDO_LEVEL=13		SDO_LEVEL=14		SDO_LEVEL=13, SDO_NUMTILES=8		SDO_LEVEL=14, SDO_NUMTILES=8	
	# Rows	Time	# Rows	Time	# Rows	Time	# Rows	Time
0.25	138	0.11	138	0.11	25	0.15	25	0.09
0.50	138	0.11	138	0.11	76	0.17	76	0.11
1.00	382	0.20	265	0.16	257	0.24	257	0.18
2.00	781	0.35	566	0.27	706	0.42	556	0.33
5.00	2333	0.92	2216	0.88	2333	1.22	2216	1.15
10.00	6078	2.34	5832	2.25	6018	3.14	5832	3.42
30.00	12303	5.51	12161	5.72	12303	7.23	12161	7.39
50.00	14857	7.59	14857	7.44	14857	9.65	14857	8.36
100.00	23982	13.48	23982	13.55	23982	15.07	23982	14.97

Block Groups Relate Times

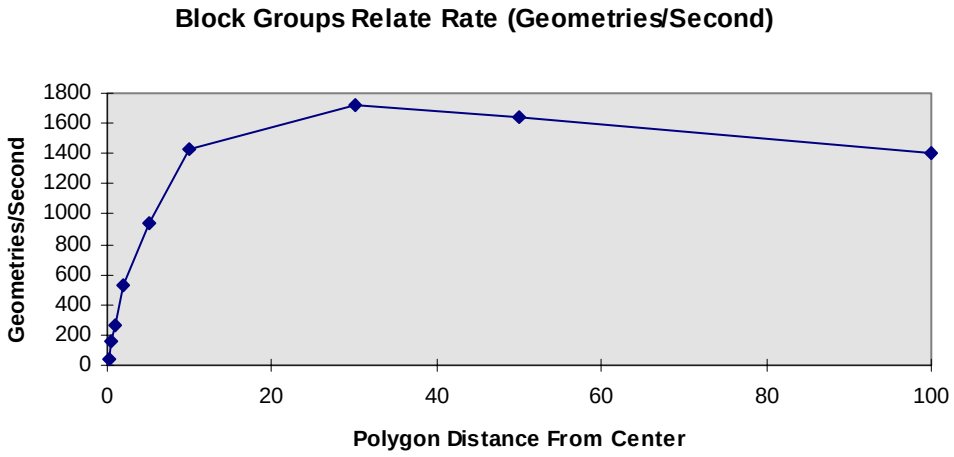
The following table shows the relate times (in seconds) for the block groups data.

Block Groups Relate Times (Seconds)

Query window	Number of rows	SDO_LEVEL=13	SDO_LEVEL=13, SDO_NUMTILES=8	SDO_LEVEL=14	SDO_ LEVEL=14, SDO_NUMTILES=8
0.25	12	0.26	0.18	0.26	0.13
0.50	45	0.28	0.28	0.28	0.22
1.00	136	0.65	0.57	0.50	0.53
2.00	359	1.22	1.12	0.67	0.70
5.00	1791	2.73	3.05	1.90	2.17
10.00	5398	5.12	5.73	3.73	4.45
30.00	12019	6.80	8.21	6.94	8.39
50.00	14682	8.89	10.57	8.95	10.54
100.00	23621	16.44	18.99	16.63	19.23

The relate rate graph, shown in Figure 17, looks very similar to the ABI data relate rate:

Figure 17: Relate Rate (BG Data)



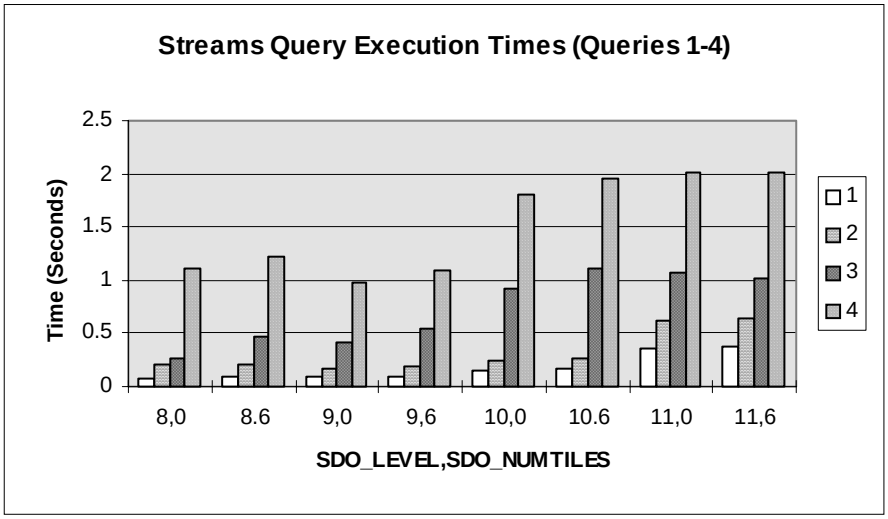
The degradation in relate rate performance is probably a function of two major factors:

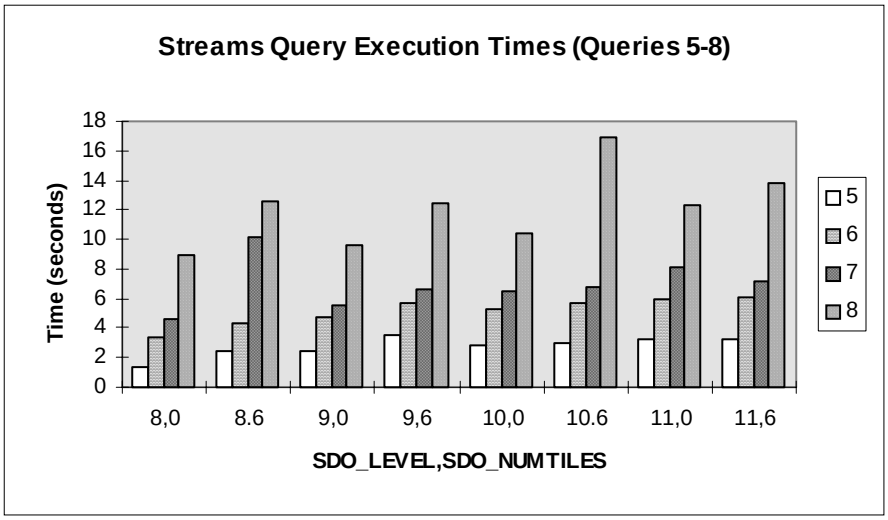
- The Block Groups data is less dense as the distance from central Manhattan increases, so a smaller proportion of the geometries can pass the secondary filter using the interior tile optimization.
- Because the geometries are more complex as they get farther from central Manhattan, the relate operation requires more time and resources.

Streams Data

Eight tests were defined for the California Streams using four point-query windows. The tests were designed to return varying amounts of data, up to about 20 percent of the total data. Queries 1, 3, 5, and 7 are SDO_FILTER operations, and queries 2, 4, 6, and 8 are SDO_RELATE operations. The query windows are the same for each pair (1 and 2, 3 and 4, 5 and 6, 7 and 8). Figure 18 shows the Streams query test results in times (seconds)

Figure 18: Streams Data Filter and Relate Query Results (Times)



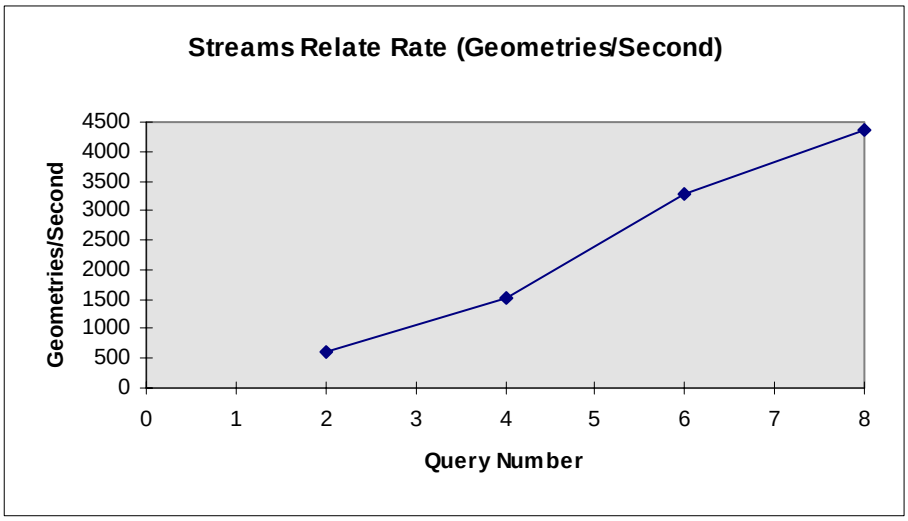


For nearly all of the tests, the fixed index at SDO_LEVEL=8 results in the best performance.

The Streams data is spread out across California, and has a more geographically random distribution than the Block Groups or ABI data. A very large amount of Block Groups and ABI data is found within 100 miles of central Manhattan (10%), and the data gets more sparse as the distance from central Manhattan increases. The California Streams data is very regular, and the relate rate continues to increase even as the query window becomes

very large. For Block Groups and ABI data, the relate rates decrease due in part to the sparseness of the data as the distance increases. Figure 19 shows the Streams relate query (query numbers 2, 4, 6, and 8) test results in geometries per second.

Figure 19: Streams Relate Query Results (Geometries/Second)



APPENDIX A - QUERIES TO DETERMINE SPACE REQUIREMENTS OF ORACLE SPATIAL

ORACLE SPATIAL TABLES

This first query can be used to find the amount of data in the base Oracle Spatial table:

```
select sum(bytes/1024/1024) TABLE_SIZE_MB
from user_extents
where segment_name = 'YOUR_SPATIAL_TABLE_NAME';
```

To find the number of extents and the amount of data in each extent in the base Oracle Spatial table the following query can be used (to determine how efficiently data is being stored in the table):

```
select bytes/1024/1024 EXTENT_SIZE_MB, segment_name, extent_id
from user_extents
```

```
where segment_name = 'YOUR_SPATIAL_TABLE_NAME'
order by segment_name, extent_id;
```

Finding the data in the lob segments associated with the table is a two-step process:

```
select segment_name from user_lobs where table_name =
     'YOUR_SPATIAL_TABLE_NAME';
```

Then for each of the segments (there should be at least two associated with each column of type SDO_GEOMETRY, and possibly more depending on the other data stored in the table: one for the SDO_ELEM_INFO_ARRAY and one for the SDO_ORDINATE_ARRAY) that came back from the previous query:

```
select sum(bytes/1024/1024) LOB_SIZE_MB
from user_extents
where segment_name = 'EACH_LOB_FROM_PREVIOUS_QUERY';
```

To find the number of extents and the amount of data in each extent associated with the LOB storage, the following query can be used (to determine how efficiently LOB data is being stored for the table):

```
select bytes/1024/1024 LOB_EXTENT_MB, extent_id
from user_extents
where segment_name =   'EACH_LOB_FROM_PREVIOUS_QUERY'
order by extent_id;
```

ORACLE SPATIAL INDEXES

First, find the name of the index table. If the index name is known, this can be done using the query:

```
select SDO_INDEX_TABLE
from all_sdo_index_metadata
where sdo_index_name = 'YOUR_SPATIAL_INDEX_NAME';
```

Finding the size of the Oracle Spatial index table can be done in a single query:

```
select sum(bytes/1024/1024) INDEX_TAB_SIZE_MB
from user_extents
where segment_name = 'YOUR_SPATIAL_IDX_TABLE_NAME';
```

To find the extent information as before, run the same query:

```
select bytes/1024/1024 EXTENT_SIZE_MB, segment_name, extent_id
from user_extents
where segment_name = 'YOUR_SPATIAL_IDX_TABLE_NAME'
order by segment_name, extent_id;
```

Next, to find the size of the indexes associated with the previous table execute the same query as above (for either the sum or the individual extents) adding an _B1 to YOUR_SPATIAL_IDX_TABLE_NAME above for the first index, and _B2 for the other index.

If you have all the information required, the entire set of size requirements can be determined in a single query:

```
select (table_bytes +lob1_bytes + lob2_bytes + index_bytes +
  b1_bytes + b2_bytes) total_bytes from
  (select sum(bytes/1024/1024) table_bytes
   from user_extents
   where segment_name='US_BG'),
   (select sum(bytes/1024/1024) lob1_bytes
   from user_extents where
   segment_name='SYS_LOB0000022570C00008$$'),
   (select sum(bytes/1024/1024) lob2_bytes
   from user_extents where
   segment_name='SYS_LOB0000022570C00009$$'),
   (select sum(bytes/1024/1024) index_bytes
   from user_extents where
   segment_name='US_BG_IDX_FL13$'),
   (select sum(bytes/1024/1024) b1_bytes
   from user_extents where
   segment_name='US_BG_IDX_FL13$_B1'),
   (select sum(bytes/1024/1024) b2_bytes
   from user_extents where
   segment_name='US_BG_IDX_FL13$_B2');
```

```
TOTAL_BYTES
-----------
  620.17031
```

APPENDIX B - EXAMPLE CONTROL FILES

The following is an example of the control file used for loading ABI point data:

```
LOAD DATA
INFILE *
TRUNCATE
CONTINUEIF NEXT(1:1) = '#'
INTO TABLE POINT
FIELDS TERMINATED BY '|'
TRAILING NULLCOLS
( GID      INTEGER EXTERNAL,
  GEOMETRY COLUMN OBJECT
    (SDO_GTYPE       INTEGER EXTERNAL,
     SDO_POINT COLUMN OBJECT
       (X            FLOAT EXTERNAL,
        Y            FLOAT EXTERNAL)
     )
)

BEGINDATA
1000000| 1| -86.032853| 39.920673|
```

An example control file for the Block Groups data set follows:

```
LOAD DATA
INFILE '/usr/load_files/bg/us_bg.dat'
INTO TABLE US_BG
replace
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
( gid    char(6),
geometry COLUMN OBJECT
( sdo_gtype INTEGER EXTERNAL,
  sdo_srid INTEGER EXTERNAL,
   isnull FILLER CHAR,
```

```
 SDO_POINT COLUMN OBJECT  NULLIF geometry.isnull="pt"
 ( X INTEGER EXTERNAL,
   Y INTEGER EXTERNAL,
   Z INTEGER EXTERNAL),
SDO_ELEM_INFO VARRAY terminated by ';'
       (SDO_ORDINATES char(38)),
SDO_ORDINATES VARRAY terminated by ':'
   (SDO_ORDINATES char(38))))
```

The following is an example row from the Block Groups data file:

```
10000,3,,pt,,,,1,3,1;-86.47625,32.46743,-86.47691,32.46864,-
86.47761,32.47007,-86.478439,32.471649,-86.479294,32.472855,-
86.4799,32.4736, (…more data …):
```

APPENDIX C - USING SPATIAL INDEX ADVISOR

Oracle Enterprise Manager includes the Spatial Index Advisor, a tool that provides a first estimate as to what the best SDO_LEVEL value will be for a given layer and includes powerful analysis functions to help determine if an optimal value for SDO_LEVEL has been chosen. Oracle suggests using the Spatial Index Advisor to get initial help in creating the index.

After creating the index, Oracle recommends using Spatial Index Advisor for visualizing the layer (or part of the layer) along with the tiles created for the index, running queries, and evaluating the selectivity of the primary filter in a query window. If the value chosen for SDO_LEVEL needs to be changed, recreate the index and iterate the process for determining whether the value for SDO_LEVEL needs to be changed.

Before the Spatial Index Advisor can be used, the USER_SDO_GEOM_METADATA view must be populated for your spatial layers. Additionally, privileges need to be granted to the user who will be using Spatial Index Advisor. A procedure to create a role and an example GRANT statement are included in Appendix D.

To determine the best index values for a table start by invoking Spatial Index Advisor:

- On UNIX platforms, type `oemapp sdoadvisor`.

- On Windows platforms, from the Start menu choose Programs, Oracle - oracle_home_name, Extended Administration, Spatial Index Advisor.

Choose the table in the Add Layer dialog by clicking on the Database, then the user, then the table. Click OK. Choose Index from the top menu bar, then Create Index (this only works if the layer is not already indexed). Click “Show SQL” on the bottom right, then click on the column of type SDO_GEOMETRY to display the Spatial tab. Enter a name for the index.

Click on the Storage tab. For spatial Indexes, Oracle recommends using an Initial and Next Size for the index table to optimize space usage, and setting Increase Size by to 0%. The amount of space to choose for Initial and Next sizes is dependent on the amount of data, the type of data, the shape represented by the data, and the tiling level. In Oracle Spatial 8.1.6 do not choose unlimited extents.

Click on the Spatial tab. This version of Spatial Index Advisor helps with creating fixed indexes, but is optimized for creating hybrid indexes. The next version of Spatial Index Advisor will include better functionality to help with fixed index creation. Since this version is geared to help build hybrid indexes, the button for fixed indexes is grayed out, but this document describes how the tool will help with fixed index creation. Click the Estimate box. The Welcome dialog for the Tiling Wizard appears and explains the process for determining the best spatial index creation parameters.

On the next page, if the spatial column of type SDO_GEOMETRY contains only point data, select point data. After that, the next page is the Sampling Analysis page. The Spatial Index Adviser will sample a percentage of the data to determine the best tiling level. By default, the percentage of rows that corresponds to 1000 rows will be used. If you specify more data to be sampled, the accuracy of the estimate is higher, but the time required to do the estimate increases.

The next page is for the Area of Interest Analysis. In the Spatial Index Advisor shipping with Oracle Spatial 8.1.6, the analysis completed is non-optimal and should not be used. The following describes the conditions where it should be used in Oracle Spatial 8.1.7

and forward: If the dimensions of a typical area of interest are known and are spatially consistent, enter the information on this page. For spatially inconsistent information, skip this page. For example, longitude/latitude dimensions are spatially inconsistent, because a one degree by one degree area of interest at the equator is very different than a one degree by degree area of interest at the north or south pole. Using area of interest analysis will allow the best SDO_LEVEL to be calculated without taking into account storage requirements. For point data, if this analysis is appropriate (the query window is known and the window is spatially consistent) the resulting SDO_LEVEL should be used. For other geometry types, this is the best SDO_LEVEL if index build time and the amount of index data generated are not important. Index build time may not be an issue at some sites; however, having too much data in the index table can be detrimental to performance.

The next page is the Tile Size Analysis. Oracle recommends using the default values - 10000 tiles for the minimum bounding rectangle encompassing all geometries in the sample. This will set up a starting point for determining the tiling level using the spatial data from the table.

The Finish page shows all the settings chosen. If all the values look correct, click Finish to complete the estimation process.

When the estimation process is complete, the Tiling Wizard shows the results of the analyses that have taken place. The first result reported is the Area Of Interest Analysis (if it was chosen, which it should not be for 8.1.6), and the value reported is the ideal value to use for SDO_LEVEL if index size does not matter. If the layer to be indexed is point data, this is the number that should be used for the index.

The next number reported is based on the Tile Size Analysis. This is the starting point for determining the tiling level based on the sampling of the spatial data.

The next value reported (Sampling - Tiling Level) is the recommended value for SDO_LEVEL on the sample from the actual data. This recommended value is determined by starting at the value calculated from the tile size analysis, then building successive

indexes at increasing SDO_LEVEL values until the index size increases dramatically. That cutoff point is the recommended SDO_LEVEL value. This is the value which will be used for spatial index creation.

The next values can be ignored unless a hybrid index is required. In most cases, hybrid indexes should not be created.

The next step is to build the index. Close the Tiling Wizard and go back to the Create Index Page. Set the Initial tiling level to the value recommended from Sampling - Tiling Level above. Set the Target number of tiles per geometry to 0 (SDO_NUMTILES) to prevent a hybrid index from being built. Also, the Commit Interval can be set to allow the index to be built with a smaller rollback segment (because it only needs to hold the number of rows specified by the commit interval), which will be useful if the index table will be very large. If the data is all point data, click on the Point Data check box. Finally, click the General tab and click Create to create the index.

After the index is created it is important to analyze the index table that was just built. To do this, use SQL*Plus. A query to build the analyze index statements associated with spatial indexes is as follows:

```
SELECT 'ANALYZE TABLE ' || sdo_index_table || ' ESTIMATE
  STATISTICS SAMPLE 1 PERCENT; ' from user_sdo_index_metadata;
```

One of these table names will begin with the index name specified on the create index statement, followed by some information about the index. Analyze this index table using the following command:

```
ANALYZE TABLE <table-name> ESTIMATE STATISTICS SAMPLE 1
PERCENT;
```

To use the Spatial Index Advisor to determine how well the index will work, some of the geometries in the layer will have to be displayed. The goal will be to have about 200 tiles for a typical area of interest. First, remove the layer (Layer->Remove Layer) then add it back again (Layer->Add Layer) so the layer can be recognized as having an index.

Next, display some geometries so that an area of interest analysis can be done. Displaying the geometries can be accomplished in one of several ways. If there aren't too many geometries in the layer the entire layer can be drawn by choosing the Layer->Draw All Geometries menu item. If the layer has a lot of data there are two choices. If there is a reference layer that can be drawn (such as a states layer or country boundaries layer) that is indexed, add the layer then display it using Layer->Draw all Geometries first. Then perform a zoom operation to a typical area of interest (by choosing View->Zoom to->Rectangle), then switch to the layer that was just indexed (from Layer Information in the lower right box, select the layer via the layer list box). Next, choose Layer->Draw Layer in Viewport. If there is no reference layer, a number of geometries in the layer can be drawn by choosing Layer->Draw Layer in Viewport->Some of the Geometries in the Layer. This allows a user definable number of geometries to be displayed.

After displaying the layer, choose Zoom to Reasonable Size from the toolbar or menu (View->Zoom To). If the spatial data shown in the viewport is close to the average expected window query size, no more tuning is required. If the viewport contains too much data (zoomed out), the tiling was done at too low a level (not enough tiles were generated) so increase SDO_LEVEL. If the viewport contains too little data (zoomed in), the SDO_LEVEL value can be safely reduced. The generated tiles can be viewed by clicking Show Fixed-size Tiles. When zooming to reasonable size, the Spatial Index Advisor zooms to where there are approximately 200 tiles in the query window.

Additional analysis can be completed by looking at actual query performance. To see how efficient the primary or secondary filter will be, first draw the layer, then click Query Rectangle. Draw a rectangle roughly equal to a query window. Choose Primary Filter Only and click OK; only the geometries associated with the primary filter are returned. If secondary filter (SDO_RELATE) operations will be executed, try Query Rectangle again, draw the same query window, choose Secondary Filter, and click on the ANYINTERACT mask. You can then visually examine the difference between the primary and secondary filter operations to determine how efficient the primary filter is.

APPENDIX D - SETUP FOR RUNNING SPATIAL INDEX ADVISOR

In order to run the Spatial Advisor tool users have to be granted certain privileges. The following creates a role which can be granted to each user who will be running Spatial Advisor. The procedure has to be run as user SYS.

```
rem
rem  NAME
rem  dbaprcre.sql
rem
rem  Copyright (c) 1999 by Oracle Corporation
rem
rem  FUNCTION
rem   Script for creating a role in the database. This role is
assigned
rem   with the least amount of privileges necessary for
running dbapps.
rem
rem  NOTES
rem   DBAs should be able to run this script against any
versions of the
rem   database, and assign this new role to any user. DBAs
must have
rem   to log in as sys in order to run this script.
rem
rem  Also, this role is likely to be part of the database in
the future.
rem     (possibly 8.1.6 or above)
rem
rem  MODIFICATION
rem     $Revision: 6 $
rem     $Author: Hsu $
rem     $Date: 5/21/99 3:37p $
rem

rem
rem Create the Role
rem
create role "OEM_DEVELOPER_ROLE"  not identified;

rem
```

```
rem This role is always needed
rem
grant "SELECT_CATALOG_ROLE" to "OEM_DEVELOPER_ROLE";

rem
rem Needed to create types, on older versions of the database,
this
rem might cause an error, which is fine.
rem
grant CREATE TYPE to "OEM_DEVELOPER_ROLE";

rem
rem This is needed for AQ.
rem
grant "AQ_ADMINISTRATOR_ROLE" to "OEM_DEVELOPER_ROLE";

rem
rem Access to these tables are needed so that we can use
faster queries, such
rem as the query used when the "tables" node is expanded in
schema manager.
rem
grant SELECT ON "SYS"."OBJ$" to "OEM_DEVELOPER_ROLE";
grant SELECT ON "SYS"."USER$" to "OEM_DEVELOPER_ROLE";

rem
rem This is needed to fetch a cluster’s hash_expression from a
v7 database.
rem
grant SELECT ON "SYS"."CDEF$" to "OEM_DEVELOPER_ROLE";

rem
rem This is needed to fetch an index’s free lists and free
list groups from a
rem v7 database. Also used for querying parallel query degree
info
rem
grant SELECT ON "SYS"."SEG$" to "OEM_DEVELOPER_ROLE";
grant SELECT ON "SYS"."IND$" to "OEM_DEVELOPER_ROLE";

rem
```

```
rem For accessing object types method pragmas info, which
doesn’t appear
rem to be available in the data dictionary.
rem
grant SELECT ON "SYS"."METHOD$" to "OEM_DEVELOPER_ROLE";

rem
rem Needed to fetch a table's cluster's schema, there’s no
easy way
rem of doing this besides selecting from sys.tab$
rem
grant SELECT ON "SYS"."TAB$" to "OEM_DEVELOPER_ROLE";

rem
rem Needed to check if an object table's type is a LOB
rem
grant SELECT ON "SYS"."TYPE$" to "OEM_DEVELOPER_ROLE";

rem
rem Needed to retrieve auto extend info in 7.2 databases
rem
grant SELECT ON "SYS"."FILEXT$" to "OEM_DEVELOPER_ROLE";

rem
rem Print a message to remind the user about logging in as sys
rem
select decode(user, 'SYS', 'Success', 'This script needs to be
run as sys!') Status from dual;
```

Next, grant the role to the Oracle Enterprise Manager user:

```
GRANT OEM_DEVELOPER_ROLE TO ORACLEUSER
```

APPENDIX E - EXAMPLE QUERIES

ABI QUERY

```
SELECT geometry
FROM abi
 WHERE MDSYS.SDO_RELATE(geometry,
```

```
    mdsys.sdo_geometry(3, null, null,
    mdsys.sdo_elem_info_array(1,3,1),
    mdsys.sdo_ordinate_array(
       -73.977989780,40.749233090,-73.978201670,40.750298410,
       -73.978818650,40.751269100,-73.979785920,40.752058880,
       -73.981017540,40.752597580,-73.982404060,40.752837320,
       -73.983822260,40.752756810,-73.985146120,40.752363190,
       -73.986257990,40.751691440,-73.987059060,40.750801270,
       -73.987478170,40.749771760,-73.987478090,40.748694410,
       -73.987058860,40.747664940,-73.986257730,40.746774820,
       -73.985145900,40.746103130,-73.983822150,40.745709550,
       -73.982404100,40.745629040,-73.981017710,40.745868770,
       -73.979786170,40.746407420,-73.978818890,40.747197140,
       -73.978201810,40.748167780,-
73.977989780,40.749233090)),
       'MASK=ANYINTERACT LAYER_GTYPE=POINT QUERYTYPE=WINDOW')
= 'TRUE';
```

Note the use of LAYER_GTYPE=POINT on the query, which provides an additional optimization for SDO_FILTER and SDO_RELATE operations not available to other geometry types.

BLOCK GROUPS QUERY

```
SELECT geometry
FROM bg
WHERE MDSYS.SDO_RELATE(geometry,
    mdsys.sdo_geometry (3, null, null,
    mdsys.sdo_elem_info_array(1,3,1),
    mdsys.sdo_ordinate_array(
      -73.977989780,40.749233090,-73.978201670,40.750298410,
      -73.978818650,40.751269100,-73.979785920,40.752058880,
      -73.981017540,40.752597580,-73.982404060,40.752837320,
      -73.983822260,40.752756810,-73.985146120,40.752363190,
      -73.986257990,40.751691440,-73.987059060,40.750801270,
      -73.987478170,40.749771760,-73.987478090,40.748694410,
      -73.987058860,40.747664940,-73.986257730,40.746774820,
      -73.985145900,40.746103130,-73.983822150,40.745709550,
      -73.982404100,40.745629040,-73.981017710,40.745868770,
      -73.979786170,40.746407420,-73.978818890,40.747197140,
```

```
        -73.978201810,40.748167780,-73.977989780,40.749233090)),
        'MASK=ANYINTERACT QUERYTYPE=WINDOW')  = 'TRUE';
```

STREAMS QUERY

```
select count(geometry)
from streams_hy
where
   mdsys.sdo_filter(geometry,mdsys.sdo_geometry(3,NULL,NULL,
                        mdsys.sdo_elem_info_array(1,3,3),
                        mdsys.sdo_ordinate_array(-
1635594.8959,
                                                 -158100.8447,
                                                 -
1610584.4854,
                                                 -
140212.6041)),
                    'querytype=window') = 'TRUE';
```

APPENDIX F - PERFORMANCE CHECKLIST AND HINTS

In Oracle Spatial there are several things to look at if a performance problem is suspected.

- Make sure all index tables have been analyzed
- Check to see if OCI_THREADED is specified for the application, and if it is, evaluate the need for having the application thread-safe.
- If the application inserts data into the Oracle Spatial object type, use bind variables rather than building the entire statement with no bind variables.
- Functions as the second argument of a spatial operator should be pulled out into an in-line view and used in conjunction with the ordered and nomerge hint. For example, the following query finds all of the ABI point data that has any interaction with the union of the two block groups polygons specified, and runs slowly:

```
SELECT d.gid
FROM us_bg a, us_bg b, user_sdo_geom_metadata c,
     abi d
WHERE a.gid=120411 and b.gid=120412 AND
```

```
c.table_name='US_BG' AND c.column_name='GEOMETRY' AND
SDO_RELATE(d.geometry, SDO_GEOM.SDO_POLY_UNION(a.geometry,
c.diminfo, b.geometry, c.diminfo),'MASK=ANYINTERACT
QUERYTYPE=WINDOW')='TRUE';
```

To make the query run faster, pull the function out of the operator and into an in-line view, using the ordered and nomerge hints:

```
SELECT /*+ ORDERED */ d.gid
FROM
  (SELECT /*+ NO_MERGE */ SDO_GEOM.SDO_POLY_UNION (a.geometry,
        c.diminfo, b.geometry, c.diminfo) GEOM_UNION
   FROM us_bg a, us_bg b, user_sdo_geom_metadata c
   WHERE a.gid=120411 and b.gid=120412 AND c.table_name =
        'US_BG' AND c.column_name='GEOMETRY') e,
  abi d
WHERE
  SDO_RELATE(d.geometry,e.geom_union, 'MASK=ANYINTERACT
  QUERYTYPE=WINDOW')='TRUE';
```

- If there is more than a single geometry specified for the window layer, include an ordered hint in the select clause and order the tables in the from clause (the table the windows come from is the first table, and the column constraining by needs to be indexed). For example:

```
SELECT a.gid
FROM abi_81_fx a, us_bg_81_fx b
WHERE (b.gid=120411 or b.gid=120412) AND
   SDO_RELATE(a.geometry, b.geometry,
   'MASK=ANYINTERACT QUERYTYPE=WINDOW')='TRUE';
```

should be changed to:

```
SELECT /*+ ORDERED */ a.gid
FROM us_bg_81_fx b, abi_81_fx a
WHERE (b.gid=120411 or b.gid=120412) AND
   SDO_RELATE(a.geometry, b.geometry,
   'MASK=ANYINTERACT QUERYTYPE=WINDOW')='TRUE';
```

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

Worldwide Inquiries:
+1.650.506.7000
Fax +1.650.506.7200
http://www.oracle.com/