Oracle Spatial Partitioning: Best Practices

An Oracle White Paper
November, 2004

ORACLE

Oracle Spatial Partitioning: Best Practices

Oracle Spatial Partitioning: Best Practices

INTRODUCTION

Oracle's scalable database architecture includes partitioning, in which a single logical table and its indexes are broken up into one or more physical tables, each with its own index. There are many different ways to partition data, however this paper will focus on range partitioning with Oracle Spatial. Range partitioning is the currently the only partitioning scheme used with spatial indexes.

This paper will discuss range partitioning, the best practices for building and maintaining partitioned spatial indexes using Oracle Spatial, and how to take advantage of a partitioning key when querying spatial data that has a partitioned spatial index.

Additionally, this paper includes new information about a novel feature of partitioned spatial indexing included in Oracle Spatial 10*g* that can significantly reduce query time and enhance scalability without requiring the specification of a partition key.

PARTITIONING: PERFORMANCE, SCALABILITY, AND MANAGEABILITY

Performance, scalability, and manageability are critical in any computing environment. Oracle has many features that enhance its utility in these areas. One of the most important Oracle features to address these requirements is partitioning.

Spatial solutions are no exception when it comes to the importance of having a high performance, scalable, and manageable solution. In fact, because spatial types may contain large volumes of data, the importance of performance, scalability, and manageability can be amplified over the more common cases of using traditional scalar data types.

This paper describes the best practices based upon real-world customer experiences for using Oracle Spatial in a partitioned environment.

Partitioning

Partitioning is an Oracle Database feature in which a single table is decomposed into multiple smaller tables called partitions. Queries do not have to be modified in any

way to use partitioning. DDL operations such as rebuilding indexes can be done against single partitions in the logical table structure rather than the whole table as a monolithic entity. This enhances the manageability of data by dividing large tasks into small tasks, and allowing for recovery should a problem occur during these DDL operations.

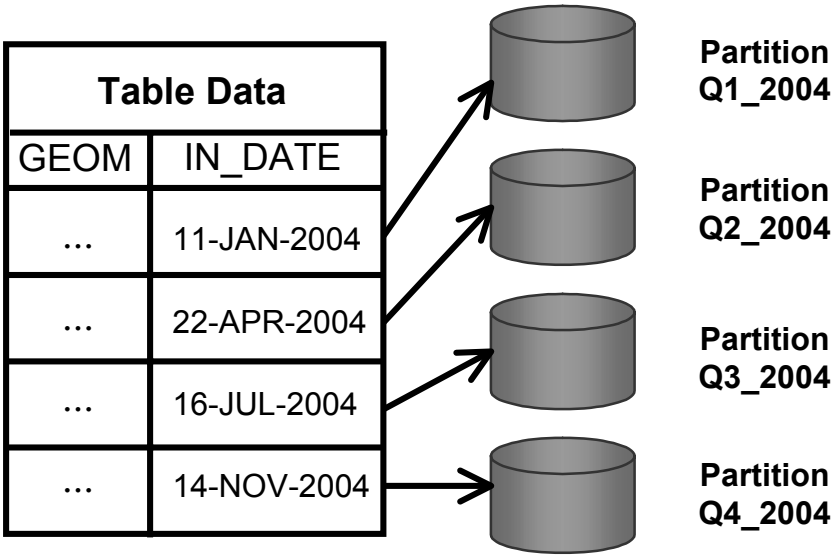
There are many different partitioning mechanisms supported in Oracle, however this paper will focus on range partitioning. Range partitioning is the supported partitioning scheme for tables with spatial indexes.

Range Partitioning in Oracle

How does Oracle know how to decompose the single logical table into multiple smaller tables? The answer to this fundamental question is that the user or DBA establishes the criterion used by the database for partitioning. In the case of range partitioning, a *partition key* is specified. The partition key is based on one or more columns in the table that are associated with a value or range of values. Oracle uses the partition key to decide which partition (smaller table) to put each row in. For example, if the partition key is a date field, then Oracle will look at the date associated with the partition key, and insert that row into the smaller table that contains the range of dates associated with the given date. For instance, given this table:

```
CREATE TABLE partn_table (in_date DATE,
  geom SDO_GEOMETRY)
PARTITION BY RANGE (in_date)
  (PARTITION Q1_2004 VALUES LESS THAN ('01-APR-2004'),
   PARTITION Q2_2004 VALUES LESS THAN ('01-JUL-2004'),
   PARTITION Q3_2004 VALUES LESS THAN ('01-OCT-2004'),
   PARTITION Q4_2004 VALUES LESS THAN ('01-JAN-2005')
  );
```

The data is loaded as follows:



In the above example, Oracle looks at the contents of the IN_DATE field and based on that value chooses which partition to put the data into. Range partitioning can also be done using a multicolumn partitioning key, where multiple column values are used to determine which partition the data goes into. This will be highlighted in a future example.

How Partitioning Enhances Manageability, Performance, and Scalability

Partitioning is used in data management solutions to address issues of manageability, scalability, and performance.

Manageability

There are many meaningful benefits of partitioning to enhance manageability in a DBMS solution.

These benefits include the ability to:

- Backup data on a partition-by-partition basis
- Restore data on a partition-by-partition basis
- Load a partition at a time
- Store data in different tablespaces on a partition-by-partition basis
- Rebuild indexes on a partition-by-partition basis
- Store indexes in different tablespaces on a partition-by-partition basis
- Merge and split partitions to consolidate or further break apart partitions
- Exchange partitions with tables
- Exchange partitions and their indexes with tables and their indexes

There are additional manageability benefits to using partitioning, which we will see as the best practices discussion begins.

Performance and Scalability

Performance and scalability are often tied together in a data management solution. If a user can do more, faster, it not only enhances the user's performance but it may allow more people to get more work done in a given amount of time.

Some of the ways scalability and performance is enhanced with the use of partitioning include Oracle's ability to:

- Search multiple table or index partitions in parallel

- Spread the I/O load associated with table or index accesses across multiple controllers
- Store data physically on disk so that data that is likely to be processed together is close together on disk
- Eliminate partitions from consideration based on a partition key

One of the most important ways partitioning can enhance performance and scalability is through the last bullet above: partition elimination.

Partition Elimination

An example of partition elimination using the previous illustration is:

```
SELECT id
FROM table
WHERE in_date = '11-JUN-2004';
```

In this case, only Partition Q2 is searched.

Partition elimination is the automatic exclusion by Oracle of partitions that will not be participating in a query. If a query includes a partition key as a predicate in the WHERE clause, Oracle will automatically route the query to the partition or partitions that are associated with the query, eliminating (and not searching) those partitions that will not have data included in the result set.

By performing partition elimination, performance and scalability are enhanced by:

- Significantly reducing the amount of table data searched to return results
- Significantly reducing the amount of index information searched

The benefits of partition elimination at query time are very significant with respect to scalability and performance.

ORACLE SPATIAL BEST PRACTICES

Oracle Spatial best practices in the area of partitioning can be broken up into two categories:

- Choosing a partitioning key
- Creating and maintaining partitioned spatial indexes

The section about choosing a partition key will follow the discussion about best practices to create and maintain partitioned spatial indexes.

Creating and Maintaining Partitioned Spatial Indexes

Users of Oracle Spatial often have hundreds or even thousands of partitions. Experience has shown that some methodologies are better than others when creating partitioned spatial indexes on tables with many partitions. When creating partitioned spatial indexes, index placement and storage parameters should be specified in the same way users would specify these parameters independent of the methodology to be shown.

The best practices steps to create a partitioned spatial index are:

1. Create the partitioned spatial index specifying the keyword UNUSABLE
2. Use ALTER INDEX REBUILD to create all of the index partitions

Create the Partitioned Index Using the UNUSABLE Keyword

Normally, when creating a partitioned index if there is any problem with any partition of the index, the entire index is invalid and must be rebuilt. There are a number of reasons one or more spatial index partitions may fail to build including a lack of physical disk space, invalid geometry types, or invalid or missing spatial reference system ID (SRID) values. If the index is created UNUSABLE, and each partition is rebuilt separately, then if one or more partition's indexes fail to build correctly only those partition's indexes will need to be rebuilt.

The following is an example of creating a partitioned spatial index using the UNUSABLE keyword:

```
CREATE INDEX partn_table_sidx ON partn_table(geom)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX
LOCAL (
  PARTITION P1 PARAMETERS (TABLESPACE='P1_TBS'),
  PARTITION P2 PARAMETERS (TABLESPACE='P2_TBS'),
  PARTITION P3 PARAMETERS (TABLESPACE='P3_TBS'),
  PARTITION P4 PARAMETERS (TABLESPACE='P4_TBS'))
UNUSABLE;
```

The LOCAL keyword above ensures a partitioned spatial index will be created, and the UNUSABLE syntax causes the index infrastructure to be created, but the local partitioned indexes are not yet built.

Rebuild each Partition's Index

After creating the index UNUSABLE, each partition's spatial index can then be created using the ALTER INDEX … REBUILD PARTITION command. If the user desires, these commands can be run from multiple sessions to allow for fine-grained user control of parallelism during the index creation process.

If during the spatial index rebuild a single partition's index fails to build for any reason, only that partition's index needs to be rebuilt. The commands to build each partition's spatial index in the above example are:

```
ALTER INDEX partn_table_sidx REBUILD PARTITION P1;
ALTER INDEX partn_table_sidx REBUILD PARTITION P2;
ALTER INDEX partn_table_sidx REBUILD PARTITION P3;
ALTER INDEX partn_table_sidx REBUILD PARTITION P4;
```

Building the index in this way not only allows for better error recovery (resulting in higher availability), it also allows queries against partitions whose indexes are USABLE to be completed. For instance, a query that includes a partition key such

that only partitions whose indexes are USABLE are accessed will return successfully even if all index partitions have no been successfully rebuilt.

In an Oracle RAC environment, the job scheduling package DBMS_SCHEDULER can be used to set up the job of rebuilding all index partitions in parallel across the RAC environment.

Maintaining Partitioned Spatial Indexes and Availability

Many customers using Oracle with spatial data want to keep all of their data on-line and available. Oracle provides the ability for partitions to be exchanged with tables, including tables that have been spatially indexed. Using EXCHANGE PARTITION, indexed spatial data in a partitioned environment remains available for general queries that do not include the partition key, and the new partition and index exchanged in requires only metadata updates in Oracle's dictionary tables. For example, assume a table is partitioned by date where each date has its own partition. Every day, a new partition's data is added. While the partition is created, loaded, and indexed, the table may be unavailable for generic processing. Again, queries that only access partitions that are usable can still access data, but non-partition specific queries against all partitions are not allowed. EXCHANGE PARTITION will keep the data available except for the brief metadata update time required to change metadata. The process for exchanging partitions is as follows:

- Create a table to hold the new data.
- Load the data into the new table
- Create the spatial index
- Create a new empty partition using the following syntax:

  ```
  ALTER TABLE partn_table ADD PARTITION q1_2005
  VALUES LESS THAN ('01-APR-2005');
  ```

  Note the ADD PARTITION syntax can only be used if the highest partition is not specified with MAXVALUE.

  A USABLE spatial index is created on the new partition automatically.

- EXCHANGE the new table and its index into the partitioned table using the ALTER TABLE … EXCHANGE PARTITION syntax

  ```
  ALTER TABLE add_partn_table

  EXCHANGE PARTITION q1_2005

  WITH TABLE new_partition

  INCLUDING INDEXES;
  ```

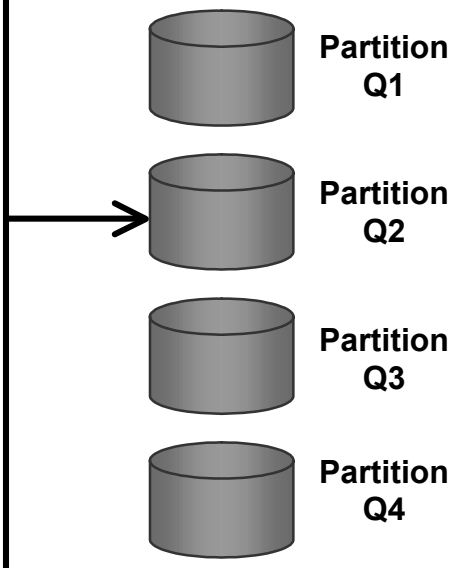
Choosing a Partitioning Key

Spatial indexes are built using Oracle's extensible indexing framework. Currently, range partitioning is the only supported partitioning mechanism in Oracle that is supported in this framework. This section will discuss how to choose a partitioning key both for the typical Oracle usage, but additionally for use with Oracle Spatial's new partition pruning mechanism.

Partitioning Key in the WHERE Clause

The most natural partitioning key to use in Oracle is a key that will often be used as a predicate in the WHERE clause of queries. For instance, if queries on a table often include a DATE field in the where clause, and then partitioning based on the DATE field would make sense. Then whenever an application accessed the data and included a DATE, Oracle would use partition pruning to only access the partitions necessary for the result.

```
SELECT in_date
FROM partn_table
WHERE sdo_anyinteract(geom,
  SDO_GEOMETRY(2003,8307,null,
    SDO_ELEM_INFO_ARRAY(1,3,3),
    SDO_ORDINATE_ARRAY(
      -75.20143, 35.80191,
      -75.16199, 35.95017))) ='TRUE'
  AND in_date > '31-MAR-2004'
  AND in_date < '01-JUL-2004';
```



Another example is partitioning by a category. If the table includes data associated with many categories, but most queries only access data referencing a specific category in the where clause, then partitioning by category would allow partition pruning to occur, so only partitions that participate in the result set would be accessed.

Spatial Partitioning

Spatial partitioning of data is accomplished by grouping data into partitions based on location. Users can partition based on X (Longitude) values, Y (Latitude) values, a combination of X and Y values, or incorporate X and Y values into an existing range partitioning key (as in the example above, to create a composite partitioning key). Spatial partitioning can done both when incorporating a partition key into queries (so the Oracle optimizer can do partition pruning), and can also be *used as a standalone partitioning mechanism even when a partition key is not specified in the query*. When incorporating spatial partitioning, spatial partition pruning can be accomplished. Spatial partition pruning is a new concept, and has not previously been described.

Spatial Partition Pruning

We've already seen that the Oracle optimizer can eliminate partitions from being searched based on a partitioning key. Spatial partition pruning is similar to the pruning the optimizer does, but it is based on location and does not require a partitioning key to be specified on the input query. When a user does a spatial query using SDO_FILTER, SDO_RELATE, or SDO_WITHIN_DISTANCE, a query window specifies an area-of-interest. Normally, the minimum bounding rectangle around that area-of-interest is compared with the minimum bounding rectangles associated with geometries in a layer using fast searches into the spatial index. When a partitioned spatial index is used, each partition's index metadata includes a column called SDO_ROOT_MBR, which contains the minimum bounding rectangle around all of the data in that index partition. At query time, Spatial indexing examines the SDO_ROOT_MBR column of each partition, and if the MBR of the area-of-interest does not overlap the SDO_ROOT_MBR, spatial partition pruning will occur and the index associated with that partition will never be searched.

Spatial Partitioning Based on X or Y value

Spatial partitioning is partitioning based on location. The simplest version of partitioning by location is based on partitioning by X (Longitude) or Y (Latitude) values.

In order to partition based on an X or Y value, a numeric column should be created in the table to hold the key value. The previous create table statement would be changed as follows:

```
CREATE TABLE new_partn_table (in_date DATE,
  geom SDO_GEOMETRY, x_value NUMBER)
PARTITION BY RANGE (X_VALUE)
  (PARTITION P_LT_90W VALUES LESS THAN (-90),
```

```
 PARTITION P_LT_0 VALUES LESS THAN (0),
 PARTITION P_LT_90E VALUES LESS THAN (90),
 PARTITION P_MAX VALUES LESS THAN (MAXVALUE)
);
```

This table shows what partition data would go into, based on the stored X_VALUE:

X VALUE	Partition
-180 <= x < -90	P_LT_90W
-90 <= x < 0	P_LT_0
0 <= x < 90	P_LT_90E
90 <= x <= 180	P_MAX

The X or Y value can be populated based on a number of algorithms, based on the spatial data type. If the spatial data is point data stored in the SDO_POINT field, the X or Y value can be populated directly from the SDO_POINT attribute of the SDO_GEOMETRY type. For example:

```
INSERT INTO new_partn_table NOLOGGING
  SELECT x.in_date,
         x.geom,
         x.geom.sdo_point.x
  FROM (SELECT in_date,
               geom
        FROM partn_table) x;
```

If the data polygon date, the X_VALUE field can be populated from the results of an SDO_GEOM.SDO_CENTROID function.

If the spatial data can be any of point, linestring, or polygon then the X_VALUE field can be populated from the results of the SDO_GEOM.SDO_POINTONSURFACE function. For example:

```
INSERT INTO new_partn_table NOLOGGING
SELECT x.in_date,
       x.geom,
       x.pt.sdo_point.x
FROM (SELECT in_date,
             geom,
             SDO_GEOM.SDO_POINTONSURFACE(geom, 0.05) pt
      FROM partn_table) x;
```

After loading the table, in this simple spatial partitioning case, any spatial operator that accesses NEW_PARTN_TABLE without specifying a partition key (which would be the normal expected use) will first access the metadata associated with each partition. Part of the metadata includes the minimum bounding rectangle around all of the geometries in each partition (stored in SDO_ROOT_MBR). If there is no interaction between the query window and the data stored in the SDO_ROOT_MBR column, then that partition will not be searched.

NOTE: Because non-point data may be associated with more than one partition, it is important to note that the partition key should not be specified in the WHERE clause or spatial queries. If the partition key is specified, the Oracle optimizer may wrongly exclude partitions from participating in a query.

Multicolumn Spatial Partitioning Based on X and Y Values

In the previous example, we showed how partitioning on X (or Y) can reduce the amount of data looked at when processing a spatial query. In that example, by only partitioning on X, we incorporated all of the Y values in each of the ranges of X. Thinking in terms of longitude and latitude, this is equivalent to including data from all the way up in the North Pole all the way down to the South Pole in each of the partitions. It may be better to partition additionally on the Y value to further partition the data.

Using multicolumn partitioning keys in range partitioned tables requires understanding the rules associated with multicolumn partitioning keys (ref: *Oracle Database Administrator's Guide, Chapter 16 - Managing Partitioned Tables and Indexes, Creating Partitioned Tables, Using Multicolumn Partitioning Keys*):

- The second key is only used if there is an exact match (by value) with the first key. Subsequent keys are only used if there are exact matches (by value) to all previous keys. What this means in practice is that range partitions can only be separated by a single value for all but the last partitioning key.

- If a single column partition key is used, the range partition goes up to but excludes the high value in the range partition. When a multicolumn partitioning key is used, for all but the last column the partitioning key includes the value specified in the range partition key.

The following example shows the creation of a table with a multicolumn range partition key (using X, then Y). Note that the X partition has to be a single value incremented by one for successive X partitions in order for the Y partition to be used. Since we are creating four X partitions, and each one of those will have four Y partitions, there will be 16 total partitions created.

```
CREATE TABLE multi_partn_table (in_date DATE,
  geom SDO_GEOMETRY, x_value NUMBER, y_value NUMBER)
PARTITION BY RANGE (X_VALUE,Y_VALUE)
  (
   PARTITION P_LT_90W_45S VALUES LESS THAN (1,-45),
   PARTITION P_LT_90W_0 VALUES LESS THAN (1,0),
   PARTITION P_LT_90W_45N VALUES LESS THAN (1,45),
   PARTITION P_LT_90W_90N VALUES LESS THAN (1,MAXVALUE),
   PARTITION P_LT_0_45S VALUES LESS THAN (2,-45),
   PARTITION P_LT_0_0 VALUES LESS THAN (2,0),
   PARTITION P_LT_0_45N VALUES LESS THAN (2,45),
   PARTITION P_LT_0_90N VALUES LESS THAN (2,MAXVALUE),
   PARTITION P_LT_90E_45S VALUES LESS THAN (3,-45),
   PARTITION P_LT_90E_0 VALUES LESS THAN (3,0),
   PARTITION P_LT_90E_45N VALUES LESS THAN (3,45),
   PARTITION P_LT_90E_90N VALUES LESS THAN (3,MAXVALUE),
   PARTITION P_LT_180E_45S VALUES LESS THAN (4,-45),
   PARTITION P_LT_180E_0 VALUES LESS THAN (4,0),
   PARTITION P_LT_180E_45N VALUES LESS THAN (4,45),
   PARTITION P_LT_180E_90N VALUES LESS THAN (4,MAXVALUE));
```

After creating this table, values can be inserted from the original table using a procedure such as the following:

```
INSERT INTO multi_partn_table NOLOGGING
SELECT x.in_date,
       x.geom,
       CEIL(ABS(-180 - x.pt.sdo_point.x)/90),
       x.pt.sdo_point.y
FROM (SELECT in_date,
             geom,
             SDO_GEOM.SDO_POINTONSURFACE(geom, 0.05) pt
      FROM partn_table) x;
```

The previous statement loads data into the table as follows:

	X_VALUE	Y_VALUE	Partition
1	(-180 < X <= -90)	-90 <= Y < -45	P_LT_90W_45S
1		-45 <= Y < 0	P_LT_90W_0
1		0 <= Y < 45	P_LT_90W_45N
1		45 <= Y <= 90	P_LT_90W_90N
2	(-90 < X <= 0)	-90 <= Y < -45	P_LT_0_45S
2		-45 <= Y < 0	P_LT_0_0
2		0 <= Y < 45	P_LT_0_45N
2		45 <= Y <= 90	P_LT_0_90N
3	(0 < X <= 90)	-90 <= Y < -45	P_LT_90E_45S
3		-45 <= Y < 0	P_LT_90E_0
3		0 <= Y < 45	P_LT_90E_45N
3		45 <= Y <= 90	P_LT_90E_90N
4	(90 < x < =180)	-90 <= Y < -45	P_LT_180E_45S
4		-45 <= Y < 0	P_LT_180E_0
4		0 <= Y < 45	P_LT_180E_45N
4		45 <= Y <= 90	P_LT_180E_90N

With the partitioning shown, spatial operators with no partitioning key will be distributed to each partition in the table. Oracle Spatial's partition pruning will be used to compare the minimum bounding rectangle around the query window with the minimum bounding rectangle stored in SDO_ROOT_MBR column in the index metadata to determine if it is possible that data stored in each partition can be involved in the result of the spatial query. If the data cannot participate in the result set, then Oracle spatial will not do any further processing in that partition.

Multicolumn Partitioning Key in the WHERE Clause Combined With Spatial Partitioning

This same kind of spatial partition elimination can be extended to use Oracle's traditional partition elimination mechanism, then applying spatial partition elimination to further prune partitions from being processed.

Since leading keys in a multicolumn partitioning key must be single value keys in order to move to the next key, the earlier example using a DATE range for each fiscal quarter does not work. In this case a pseudo-key column must be used to ensure the range partition for the leading key is an exact value. When adding a quarterly partition to the previous example the create table statement would look like this:

```
CREATE TABLE quarter_spatial_partn_table (
   in_date DATE, quarter NUMBER,
   geom SDO_GEOMETRY, x_value NUMBER, y_value NUMBER)
PARTITION BY RANGE (QUARTER, X_VALUE, Y_VALUE)
  (
   PARTITION Q1_P_LT_90W_45S VALUES LESS THAN (1,1,-45),
   PARTITION Q1_P_LT_90W_0 VALUES LESS THAN (1,1,0),
   PARTITION Q1_P_LT_90W_45N VALUES LESS THAN (1,1,45),
   PARTITION Q1_P_LT_90W_90N VALUES LESS THAN (1,1,90),
   PARTITION Q1_P_LT_0_45S VALUES LESS THAN (1,2,-45),
   PARTITION Q1_P_LT_0_0 VALUES LESS THAN (1,2,0),
   PARTITION Q1_P_LT_0_45N VALUES LESS THAN (1,2,45),
   PARTITION Q1_P_LT_0_90N VALUES LESS THAN (1,2,90),
   PARTITION Q1_P_LT_90E_45S VALUES LESS THAN (1,3,-45),
   PARTITION Q1_P_LT_90E_0 VALUES LESS THAN (1,3,0),
   PARTITION Q1_P_LT_90E_45N VALUES LESS THAN (1,3,45),
   PARTITION Q1_P_LT_90E_90N VALUES LESS THAN (1,3,90),
   PARTITION Q1_P_LT_180E_45S VALUES LESS THAN (1,4,-45),
   PARTITION Q1_P_LT_180E_0 VALUES LESS THAN (1,4,0),
   PARTITION Q1_P_LT_180E_45N VALUES LESS THAN (1,4,45),
   PARTITION Q1_P_LT_180E_90N VALUES LESS THAN (1,4,90),
   . . . .
```

```
  PARTITION Q4_P_LT_90W_45S VALUES LESS THAN (4,1,-45),
  PARTITION Q4_P_LT_90W_0 VALUES LESS THAN (4,1,0),
  PARTITION Q4_P_LT_90W_45N VALUES LESS THAN (4,1,45),
  PARTITION Q4_P_LT_90W_90N VALUES LESS THAN (4,1,90),
  PARTITION Q4_P_LT_0_45S VALUES LESS THAN (4,2,-45),
  PARTITION Q4_P_LT_0_0 VALUES LESS THAN (4,2,0),
  PARTITION Q4_P_LT_0_45N VALUES LESS THAN (4,2,45),
  PARTITION Q4_P_LT_0_90N VALUES LESS THAN (4,2,90),
  PARTITION Q4_P_LT_90E_45S VALUES LESS THAN (4,3,-45),
  PARTITION Q4_P_LT_90E_0 VALUES LESS THAN (4,3,0),
  PARTITION Q4_P_LT_90E_45N VALUES LESS THAN (4,3,45),
  PARTITION Q4_P_LT_90E_90N VALUES LESS THAN (4,3,90),
  PARTITION Q4_P_LT_180E_45S VALUES LESS THAN (4,4,-45),
  PARTITION Q4_P_LT_180E_0 VALUES LESS THAN (4,4,0),
  PARTITION Q4_P_LT_180E_45N VALUES LESS THAN (4,4,45),
  PARTITION Q4_P_LT_180E_90N VALUES LESS THAN (4,4,90)
 );
```

To load the data from an existing table into the correct partitions, an insert statement such as the following could be run:

```
INSERT INTO quarter_spatial_partn_table NOLOGGING
SELECT x.in_date,
    ceil (months_between (x.in_date,
    TO_DATE('31-DEC-2003'))/3),
    x.geom,
    ceil(abs(-180 - x.pt.sdo_point.x)/90),
    x.pt.sdo_point.y
FROM (SELECT in_date,
             geom,
             SDO_GEOM.SDO_POINTONSURFACE(geom, 0.05) pt
      FROM partn_TABLE) x;
```

The Oracle optimizer will ensure that a spatial query on the above table that includes a partitioning key will only access the partitions associated with that partitioning key. Further, spatial will look at the index metadata associated with all partitions that are still available to satisfy the query criteria, and will only access partitions that could have data that overlaps the minimum bounding rectangle around the query window.

Other Spatial Partitioning Considerations

There are other spatial partitioning considerations that should be taken into account when deciding how/whether to incorporate spatial partitioning into a data storage solution.

Geometries that Span Multiple Spatial Partitions

Very large geometries that cross partition bounds will expand the bounds of the data in the partition, overlapping other partitions. This will expand the SDO_ROOT_MBR area, and potentially cause more partitions to be searched for any given spatial query. This is never a problem for point data, but may be an issue for lines or polygons. If there are few of these large geometries, they may be put in their own separate partition.

Cost of Spatial Partition Pruning

Spatial partition elimination is more costly than partition elimination in the optimizer. Currently, for every partition that Oracle Spatial needs to eliminate, the cost is just over 1 millisecond per partition. This testing was done on a Hewlett Packard Integrity RX4640 server running Red Hat Linux Advanced Server 3.0. This hardware configuration included 4 1.5 Ghz Itanium CPUs, 16 Gbytes of memory, and an HP Storageworks Enterprise Virtual Array 5000 Running VCS 3.010. This overhead is likely to be addressed in a future release.

CONCLUSION

Best practices when using Oracle Spatial with Partitioning include creating indexes unusable and then rebuilding each partition's index separately. Using this mechanism error recovery is much easier, and partitions that are successfully indexed can be queried. In addition, users have full control over how parallelism is used during the index build. Exchange partition can be done to quickly add new indexed partitions or to rebuild a single partition's index without have to rebuild o the entire spatial index.

Spatial partitioning can be used to locate features in the same area on disk in the same partition. Oracle's spatial partition pruning mechanism can remove partitions from consideration if the minimum bounding rectangle around the data in the partition has no interaction with the minimum bounding rectangle around the query window.

Oracle Spatial Partitioning: Best Practices
November, 2004
Author: Daniel Abugov
Contributing Authors:

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

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