Oracle8™ Spatial Cartridge

User’s Guide and Reference

Release 8.0.5

May 1998

Part No. A53264-03

Enabling the Information Age™

Oracle8 Spatial Cartridge User’s Guide and Reference

Part No. A53264-03

Release 8.0.5

Primary Author: Jeff Hebert

Contributing Author: Anna Logan

Contributors: Dan Geringer, Peter Vretanos, Jayant Sharma

Contents

Send Us Your Comments

Oracle8 Spatial Cartridge User's Guide and Reference, Release 8.0.5

Part No. A53264-03

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available).

You can send comments to us in the following ways

- e-mail: nedc_doc@us.oracle.com
- FAX: 603.897.3269. Attn: Spatial Cartridge
- postal service:
  Oracle Corporation
  Oracle Spatial Cartridge Documentation
  One Oracle Drive
  Nashua, NH 03062
  USA

If you would like a reply, please include your name, address, and telephone number.

Preface

The *Oracle8 Spatial Cartridge User's Guide and Reference* provides user and reference information for Spatial Cartridge, and extensions to Oracle8 Enterprise Edition.

Spatial Cartridge requires Oracle8 Enterprise Edition. Oracle8 and Oracle8 Enterprise Edition have the same basic features. However, several advanced features, such as data cartridges, are available only with the Enterprise Edition, and some of these features are optional. For example, to use Oracle8 table partitioning, you must have the Enterprise Edition and the Partitioning Option.

For information about the differences between Oracle8 and the Oracle8 Enterprise Edition and the features and options that are available to you, see *Getting to Know Oracle8*.

Note : Spatial Cartridge release 8.0.3 introduced two distinct algorithms for building a spatial index: fixed-size tiling and variable-sized tiling. Based on testing and customer feedback Oracle now recommends using only fixed-size tiling on production systems. Variable-sized tiling, while it has theoretical advantages in some situations, is included primarily for experimentation purposes. In the future, Oracle may issue a technical bulletin further refining the usage of variable-sized tiling.

Intended Audience

This guide is intended for anyone who needs to store spatial data in a relational database.

Structure

This guide contains nine chapters, several appendixes, and a glossary:

Chapter 1	Introduces spatial data concepts.
Chapter 2	Explains spatial data loading.
Chapter 3	Explains methods for querying a spatial database.
Chapter 4	Explains how to use partitioned point data.
Chapter 5	Provides the syntax and semantics for the administrative functions and procedures.
Chapter 6	Provides the syntax and semantics for the tuning functions and procedures.
Chapter 7	Provides the syntax and semantics for the geometric functions and procedures.
Chapter 8	Provides the syntax and semantics for the window functions and procedures.
Chapter 9	Provides the syntax and semantics for functions and procedures relevant only to using partitioned point data.
Appendix A	Describes sample SQL scripts and tuning tips.
Appendix B	Describes the Spatial Cartridge data dictionary.
Appendix C	Provides a list of error messages and conditions.
Glossary	Provides definitions of terms used in this guide.

Related Documents

For more information, see the following manuals:

- *Getting to Know Oracle8*
- *Oracle8 Administrator's Guide*
- *Oracle8 Error Messages*
- *Oracle8 Utilities*

For additional information about Spatial Cartridge, including a demonstration, several white papers, and other assorted collateral, visit the official Spatial Cartridge web site: `http://www.oracle.com/st/cartridges/spatial/`

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
%	The percent sign represents the system prompt on a UNIX system.

1 Spatial Cartridge Concepts

Oracle Spatial Cartridge is an integrated set of functions and procedures that enables spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle8 database.

Spatial data represents the essential location characteristics of real or conceptual objects as those objects relate to the real or conceptual space in which they exist.

1.1 Introduction to Spatial Data

Spatial Cartridge is designed to make the storage, retrieval, and manipulation of spatial data easier and more natural to users such as a Geographic Information System (GIS). Once this data is stored in an Oracle8 relational database, it can be easily and meaningfully manipulated and retrieved as it relates to all the other data stored in the database.

A common example of spatial data can be seen in a road map. A road map is a two-dimensional object that contains points, lines, and polygons that can represent cities, roads, and political boundaries such as states or provinces. A road map is a visualization of geographic information. The location of cities, roads, and political boundaries that exist on the surface of the Earth are projected onto a two-dimensional display or piece of paper, preserving the relative positions and relative distances of the rendered objects.

The data that indicates the Earth location (latitude and longitude, or height and depth) of these rendered objects is the spatial data. When the map is rendered, this spatial data is used to project the locations of the objects on a two-dimensional piece of paper. A GIS is often used to store, retrieve, and render this Earth-relative spatial data.

Other types of spatial data that can be stored using Spatial Cartridge besides GIS data include data from computer-aided design (CAD) and computer-aided manufacturing (CAM) systems. Instead of operating on objects on a geographic scale, CAD/CAM systems work on a smaller scale such as for an automobile engine or much smaller scale as for printed circuit boards.

The differences among these three systems are only in the scale of the data, not its complexity. They might all actually involve the same number of data points. On a geographic scale, the location of a bridge can vary by a few tenths of an inch without causing any noticeable problems to the road builders. Whereas, if the diameter of an engine's pistons are off by a few tenths of an inch, the engine will not run. A printed circuit board is likely to have many thousands of objects etched on its surface that are no bigger than the smallest detail shown on a roadbuilder's blueprints.

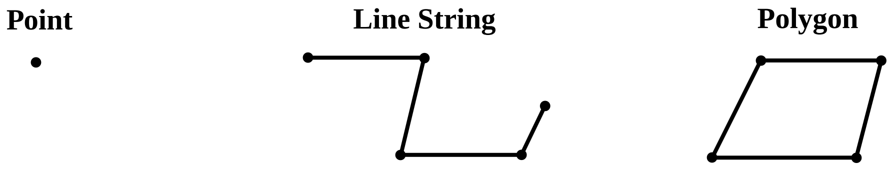
1.2 Geometric Types

Spatial Cartridge supports three geometric primitive types and geometries composed of collections of these types. The three primitive types are as follows:

- 2-D Point and Point Cluster
- 2-D Line Strings
- 2-D N-Point Polygons

2-D points are elements composed of two ordinates, X and Y, often corresponding to longitude and latitude. **Line strings** are composed of one or more pairs of points that define line segments. **Polygons** are composed of connected line strings that form a closed ring and the interior of the polygon is implied. Figure 1–1 illustrates the supported geometric primitive types.

Figure 1–1 Geometric Primitive Types



Self-crossing polygons are not supported although self-crossing line strings are. If a line string crosses itself it does not become a polygon. A self-crossing line string does not have any implied interior.

1.3 Data Model

The Spatial Cartridge data model is a hierarchical structure consisting of elements, geometries, and layers, which correspond to representations of spatial data. Layers are composed of geometries (or geometric objects), which in turn are made up of elements.

For example, a point might represent a building location, a line string might be a road or flight path, and a polygon could be a state, city, zoning district, or city block.

1.3.1 Element

An **element** is the basic building block of a geometric feature for Spatial Cartridge. The supported spatial element types are points, line strings, and polygons. For example, elements might model star constellations (point clusters), roads (line strings), and county boundaries (polygons). Each coordinate in an element is stored as an X,Y pair.

Point data[1] consists of one coordinate. **Line data** consists of two coordinates representing a line segment of the element. **Polygon data** consists of coordinate pair values, one vertex pair for each line segment of the polygon. Coordinates are defined in either a clockwise or counter-clockwise order around the polygon.

If an element spans more than one row, an incremental sequence number (starting at zero) orders the rows.

1.3.2 Geometry

A **geometry** is the representation of a user's spatial feature, modeled as an ordered set of primitive elements. Each geometric object is required to be uniquely identified by a numeric geometry identifier (GID), associating the object with its corresponding attribute set.

A complex geometric feature such as a polygon with holes would be stored as a sequence of polygon elements. In a multi-element polygonal geometry, all subelements are wholly contained within the outermost element, thus building a more complex geometry from simpler pieces.

For example, a geometry might describe the buildable land in a town. This could be represented as a polygon with holes where water or zoning prevents construction.

[1] Point data can also be stored in a partitioned table. See Chapter 4, "Partitioning Point Data" for details.

1.3.3 Layer

A **layer** is a heterogeneous collection of geometries having the same attribute set. For example, one layer in a GIS might include topographical features, while another describes population density, and a third describes the network of roads and bridges in the area (lines and points). Each layer's geometric objects and their associated spatial index are stored in the database in standard tables.

1.4 Database Structures

Spatial Cartridge uses four database tables to store and index spatial data. These four tables are collectively referred to as a *layer*. A template SQL script is provided to facilitate the creation of these tables. See Section A.1.1.2, "crlayer.sql Script" for details.

Table 1–1 through Table 1–4 describe the schema of a Spatial Cartridge layer.

Table 1–1 <layername>_SDOLAYER

SDO_ORDCNT	**SDO_LEVEL**	**SDO_NUMTILES**	**SDO_COORDSYS**
<number>	<number>	<number>	<varchar>

Table 1–2 <layername>_SDODIM table or view

SDO_DIMNUM	**SDO_LB**	**SDO_UB**	**SDO_TOLERANCE**	**SDO_DIMNAME**
<number>	<number>	<number>	<number>	<varchar>

Table 1–3 <layername>_SDOGEOM table or view

SDO_GID	**SDO_ESEQ**	**SDO_ETYPE**	**SDO_SEQ**	**SDO_X1**	**SDO_Y1**	**...**	**SDO_Xn**	**SDO_Yn**
<number>	<number>	<number>	<number>	<number>	<number>	...	<number>	<number>

Table 1–4 <layername>_SDOINDEX table

SDO_GID	**SDO_CODE**	**SDO_MAXCODE ****	**SDO_GROUPCODE ****	**SDO_META**
<number>	<raw>	<raw>	<raw>	<raw>

The SDO_MAXCODE and SDO_GROUPCODE columns are not required for the recommended indexing algorithm using fixed-size tiles.

The columns of each table are defined as follows:

<layername>_SDOLAYER:

- **SDO_ORDCNT -** The SDO_ORDCNT column is the total number of ordinates per row in the <layername>_SDOGEOM table. That is, the total number of data value columns, and not the number of points or coordinates. SDO_ORDCNT should not be multiplied by the total number of dimensions per coordinate as it is already a total.
- **SDO_LEVEL -** The SDO_LEVEL column stores the number of times the layer should be tessellated during the indexing stage. Use the SDO_TUNE.ESTIMATE_TILING_LEVEL() procedure to determine an appropriate tiling level for your data.
- **SDO_NUMTILES -** The SDO_NUMTILES column is the number of variable-sized tiles used to tessellate each object in the <layername>_SDOGEOM table. This column must be set to NULL when using fixed-size tiles.
- **SDO_COORDSYS -** The SDO_COORDSYS column is optional; where you can indicate the name of the coordinate system, using a standard such as *POSC* or *OGIS*.

<layername>_SDODIM:

- **SDO_DIMNUM -** The SDO_DIMNUM column is the dimension to which this row refers, starting with 1 and increasing.
- **SDO_LB -** The SDO_LB column is the lower bound of the ordinate in this dimension. For example, if the dimension is latitude, the lower bound would be -90.
- **SDO_UB -** The SDO_UB column is the upper bound of the ordinate in this dimension. For example, if the dimension is longitude, the upper bound would be 180.
- **SDO_TOLERANCE -** The SDO_TOLERANCE column is the distance two points can be apart and still be considered the same due to round-off errors. Tolerance must be greater than zero. If you want zero tolerance, enter a number such as 0.00005, where the number of zeroes to the right of the decimal point matches the precision of your data. The extra *5* will round up to your last decimal digit.
- **SDO_DIMNAME -** The SDO_DIMNAME column is used for the usual name applied to this dimension, such as *longitude, latitude, X* or *Y.*

<layername>_SDOGEOM:

- **`SDO_GID`** - The SDO_GID column is a unique numeric identifier for each geometry in a layer.
- **`SDO_ESEQ`** - The SDO_ESEQ column enumerates each element in a geometry, that is, the **E**lement **SEQ**uence number.
- **`SDO_ETYPE`** - The SDO_ETYPE column is the geometric primitive type of the element. For this release of Spatial Cartridge, the valid values are SDO_GEOM.POINT_TYPE, SDO_GEOM.LINESTRING_TYPE, or SDO_GEOM.POLYGON_TYPE (ETYPE values 1, 2, and 3, respectively). Setting the ETYPE to zero indicates that this element should be ignored. See Section A.2.6 for information on ETYPE 0.
- **`SDO_SEQ`** - The SDO_SEQ column records the order (the **SEQ**uence number) of each row of data making up the element.
- **`SDO_X1`** - X value of the first coordinate.
- **`SDO_Y1`** - Y value of the first coordinate.
- **`SDO_Xn`** - X value of the Nth coordinate.
- **`SDO_Yn`** - Y value of the Nth coordinate.

<layername>_SDOINDEX:

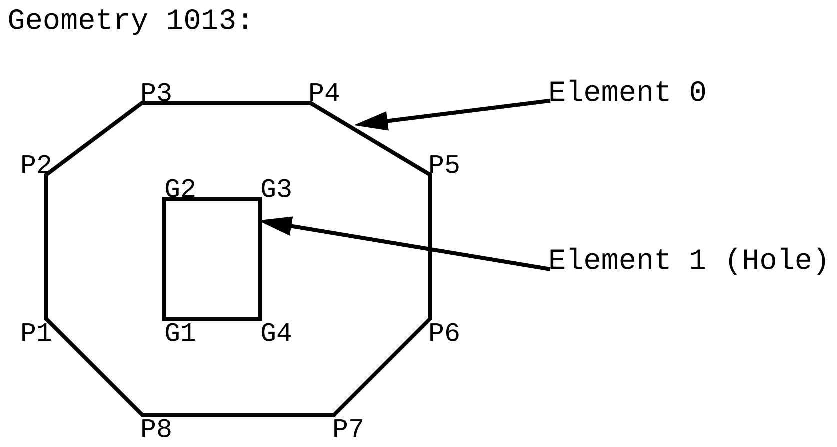
- **`SDO_GID`** - The SDO_GID column is a unique numeric identifier for each geometry in a layer. This can be thought of as a foreign key back to the <layername>_SDOGEOM table.
- **`SDO_CODE`** - The SDO_CODE column is the bit-interleaved ID of a tile that covers SDO_GID. The number of bytes needed for the SDO_CODE and SDO_MAXCODE columns depends on the level used for tiling. Use the `SDO_ADMIN.SDO_CODE_SIZE()` function to determine the size required for a given layer. The maximum number of bytes possible is 255.
- **`SDO_MAXCODE`** - The SDO_MAXCODE column describes a variable-sized logical tile, which is the smallest tile (with the longest tile ID) in the current quadrant. The SDO_MAXCODE column is SDO_CODE padded out one place farther than the longest allowable code name for this index. This column is not used for fixed-size tiles.
- **`SDO_GROUPCODE`** - The SDO_GROUPCODE column is a prefix of SDO_CODE. It represents a variable-sized tile at level <layername>_SDOLAYER.SDO_LEVEL that contains or is equal to the tile represented by SDO_CODE. This column is not used for fixed-size tiles.

- **SDO_META -** The SDO_META column is not required for spatial queries. It provides information necessary to find the bounds of a tile. See Section A.2.8 for one possible use of this column.

Spatial Cartridge provides stored procedures that assume the existence of the layer schema as described in this section. While layer tables may contain additional columns, they are required to contain at least the columns described in this section with the same column names and data types.

Figure 1–2 illustrates how a geometry is stored in the database using Spatial Cartridge. The geometry to be stored is a complex polygon with a hole in it.

Figure 1–2 Complex Polygon



<layername>_SDOLAYER

SDO_ORDCNT (number)
4

<layername>_SDODIM

SDO_DIMNUM (number)	SDO_LB (number)	SDO_UB (number)	SDO_TOLERANCE (number)	SDO_DIMNAME (varchar)
1	0	100	.05	X axis
2	0	100	.05	Y axis

<layername>_SDOGEOM

SDO_GID (number)	SDO_ESEQ (number)	SDO_ETYPE (number)	SDO_SEQ (number)	SDO_X1 (number)	SDO_Y1 (number)	SDO_X2 (number)	SDO_Y2 (number)
1013	0	3	0	P1(X)	P1(Y)	P2(X)	P2(Y)
1013	0	3	1	P2(X)	P2(Y)	P3(X)	P3(Y)
1013	0	3	2	P3(X)	P3(Y)	P4(X)	P4(Y)
1013	0	3	3	P4(X)	P4(Y)	P5(X)	P5(Y)
1013	0	3	4	P5(X)	P5(Y)	P6(X)	P6(Y)
1013	0	3	5	P6(X)	P6(Y)	P7(X)	P7(Y)
1013	0	3	6	P7(X)	P7(Y)	P8(X)	P8(Y)
1013	0	3	7	P8(X)	P8(Y)	P1(X)	P1(Y)
1013	1	3	0	G1(X)	G1(Y)	G2(X)	G2(Y)
1013	1	3	1	G2(X)	G2(Y)	G3(X)	G3(Y)
1013	1	3	2	G3(X)	G3(Y)	G4(X)	G4(Y)
1013	1	3	3	G4(X)	G4(Y)	G1(X)	G1(Y)

In this example, the <layername>_SDOGEOM table is shown as an eight column table with four ordinates per row. In actual usage, Spatial Cartridge supports N-wide[1] tables. The coordinates for the outer polygon in this example could have been loaded into a single row containing values for coordinates P1 to P8, and then repeating P1 to close the polygon. The coordinates would be stored in the SDO_X1 and SDO_Y1 through SDO_X9 and SDO_Y9 columns.

The data in the <layername>_SDOINDEX table is described in Section 1.5, "Indexing Methods".

1.5 Indexing Methods

A spatial index is considered a logical index as opposed to a physical index. The entries in the spatial index are dependent on the location of the geometric objects in the layer space, and are not dependent on the stored location of the data on the disk. This means that a table containing spatial data could be moved or split and the spatial index would not need to be rebuilt.

Spatial Cartridge release 8.03 introduced two distinct algorithms for building a spatial index: fixed-size tiling and variable-sized tiling. Based on testing and customer

[1] A <layername>_SDOGEOM table can have up to 255 columns. The maximum number of data columns is 255, minus 4 for the other required spatial columns, and minus any other user-defined columns. For polygon and line strings, storing 16 to 20 ordinates per row is suggested for performance reasons, but not required.

feedback, Oracle currently recommends using only fixed-size tiling on production systems. Variable-sized tiling, while it has theoretical advantages in some situations, is included for experimentation purposes only.

In spatial indexing, the object space (the layer where all geometric objects are located) is subjected to a process called **tessellation**, which defines exclusive and exhaustive cover tiles of every stored element. Spatial Cartridge can use either fixed-size or variable-sized tiles to cover a geometry.

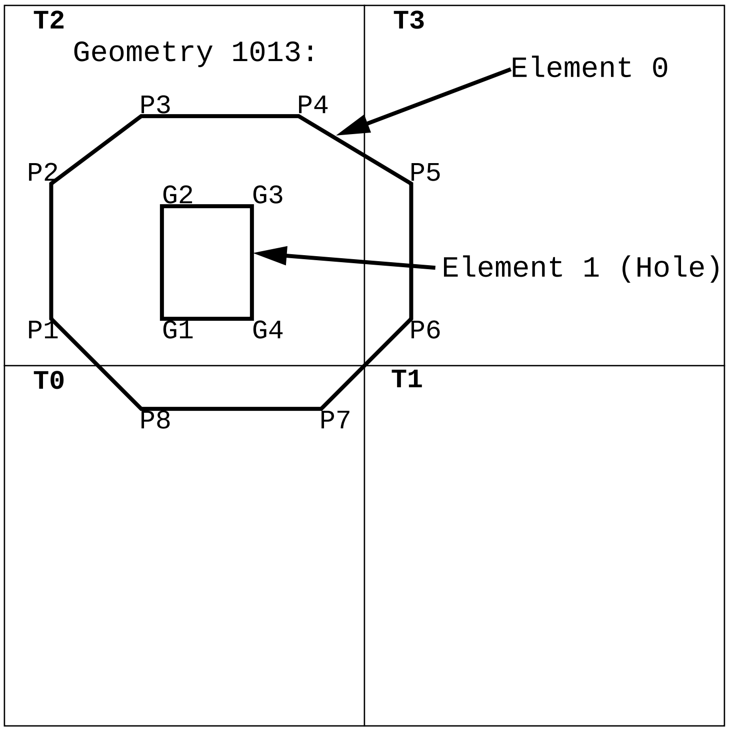
The number of tiles used to cover an element is a user-tunable parameter. Using either smaller fixed-size tiles or more variable-sized tiles provides a better fit of the tiles to the element. The fewer the number of tiles or the larger the tiles, the coarser the fit.

1.5.1 Tessellation of a Layer

The process of determining which tiles cover a given element is called tessellation. The tessellation process is a quadtree decomposition, where the object space is broken down into four equal-sized covering tiles. Successive tessellations break those tiles down into four smaller tiles, and this process continues until the desired level has been achieved. The results of the tessellation process on an element are stored in the <layername>_SDOINDEX table. See Section 2.3, “Index Creation” for more information on tessellation.

Figure 1–3 illustrates geometry 1013 tessellated to a maximum of four cover tiles. The cover tiles are then shown stored in the <layername>_SDOINDEX table.

Figure 1–3 Tessellated Figure



Only three of the four tiles generated by the first tessellation interact with the geometry. Only those tiles that interact with the geometry are stored in the <layername>_SDOINDEX table, as shown in Table 1–5. In this example, three fixed-size tiles are used.

Table 1–5 <layername>_SDOINDEX Using Fixed-Size Tiles

SDO_GID <number>	SDO_CODE <raw>
1013	T0
1013	T2
1013	T3

All elements in a geometry are tessellated. In a multi-element polygon like 1013, Element 1 is already covered by tile T2 from the tessellation of Element 0.

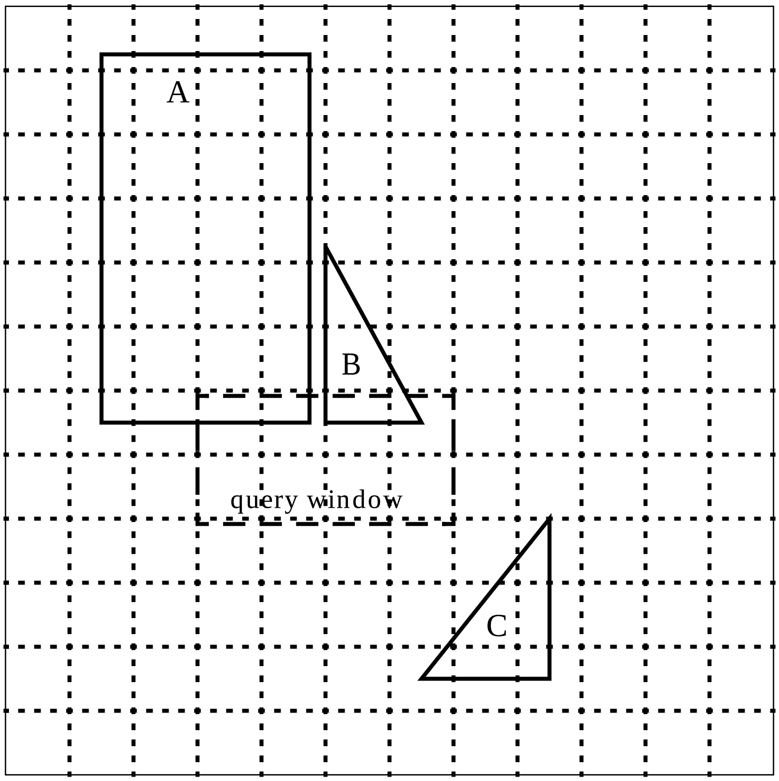
1.5.2 Fixed-Size Tile Spatial Indexing

Fixed-size tile spatial indexing is the recommended indexing method. This method uses cover tiles of equal size to cover a geometry. Because all the tiles are the same size, the standard SQL equality operator (=) can be used to compare tiles during a join operation. This results in excellent performance characteristics.

If you select a small fixed-size tile to cover small geometries and then try to use the same size tile to cover a very large geometry, a large number of tiles would be required, thereby increasing the size of the index table. However, if the fixed-size tile size chosen is large, so that fewer tiles are generated in the case of a large geometry, then the index selectivity suffers because the large tiles do not fit the small geometries very well. Figure 1–4 and Figure 1–5 illustrate the relationships between tile size, selectivity, and the number of cover tiles.

Using a small fixed-size tile as shown in Figure 1–4, selectivity is good, but a large number of tiles is needed to cover large geometries. A window query would easily identify geometries A and B, but would reject C.

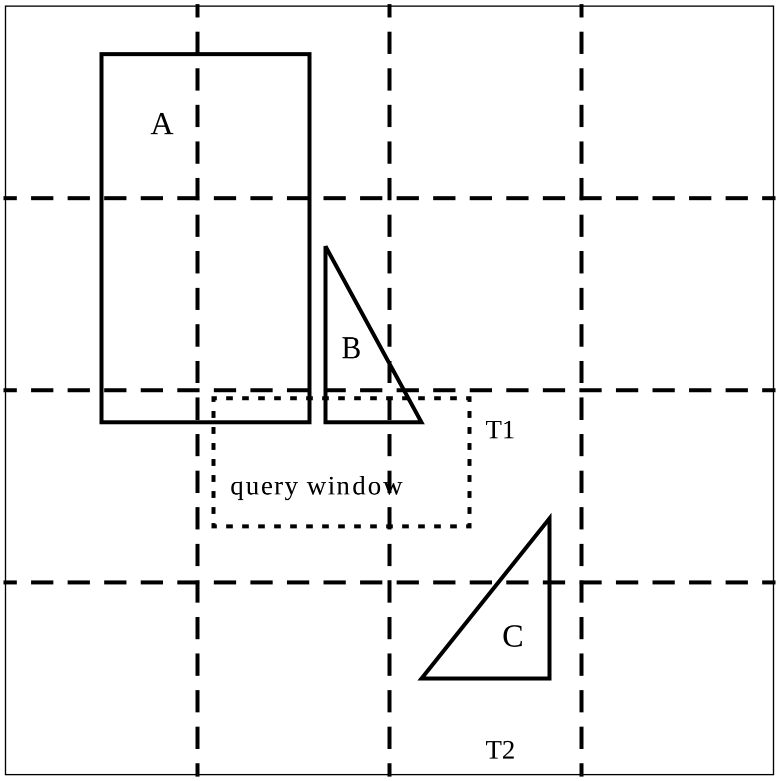
Figure 1–4 Fixed-Size Tiling with Many Small Tiles



Using a large fixed-size tile as shown in Figure 1–5, fewer tiles are needed to cover the geometries, but the selectivity is poor. A window query would likely pick up all three geometries. Any object that shares tile T1 or T2 would identify object C as a candidate, even though the objects may be far apart, such as objects B and C are in this figure.

Use the `SDO_TUNE.ESTIMATE_TILING_LEVEL()` function to determine an appropriate tiling level for your data set.

Figure 1–5 Fixed-Size Tiling with Fewer Large Tiles



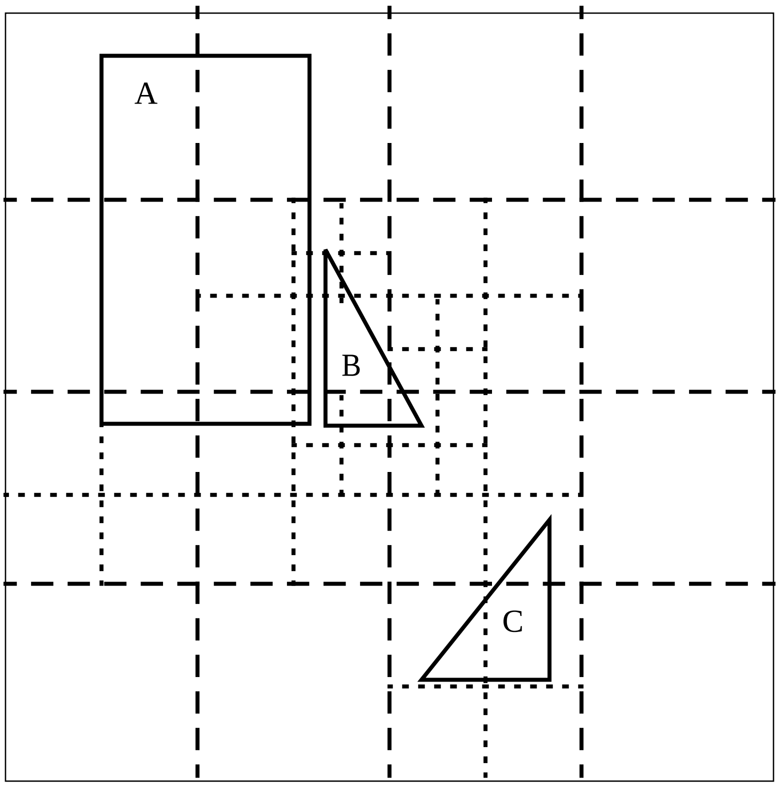
1.5.3 Variable-Sized Tile Spatial Indexing

Note: Variable-sized tile spatial indexing is not recommended for production environments. It is included primarily for experimentation purposes.

Variable-sized tile spatial indexing uses tiles of different sizes to approximate a geometry. The user specifies the number of tiles per object that should be used to approximate it and this governs the tiling process. As in the case of a linear quadtree, the cover tiles depend on the size and shape of each geometry being indexed and therefore good primary filter selectivity can be achieved. Figure 1–6 illustrates the approximation that variable-sized tiles can achieve.

In Figure 1–6, the variable-sized cover tiles conform closely to each geometry, resulting in good selectivity. The number of tiles needed to cover a geometry is controlled using the SDO_NUMTILES column in the <layername>_SDOLAYER table. See Section 2.3.3 for information on selecting appropriate values for variable-sized tiling.

Figure 1–6 Variable-Sized Tile Spatial Indexing



Two geometries may interact if a tile of one object is equal to, inside of, or contains a tile of the other. Thus, the query predicate to compare tiles involves a test for either equality or containment. This is unlike fixed-size tiling, which only requires an equality check. Example 1–1 demonstrates this feature ("5" is an arbitrary window identifier).

Example 1–1

```
SELECT r.sdo_gid
FROM roads_sdoindex r,
     window_sdoindex w
WHERE w.sdo_gid = 5
  AND (r.sdo_code BETWEEN w.sdo_code AND w.sdo_maxcode OR
       w.sdo_code BETWEEN r.sdo_code AND r.sdo_maxcode);
```

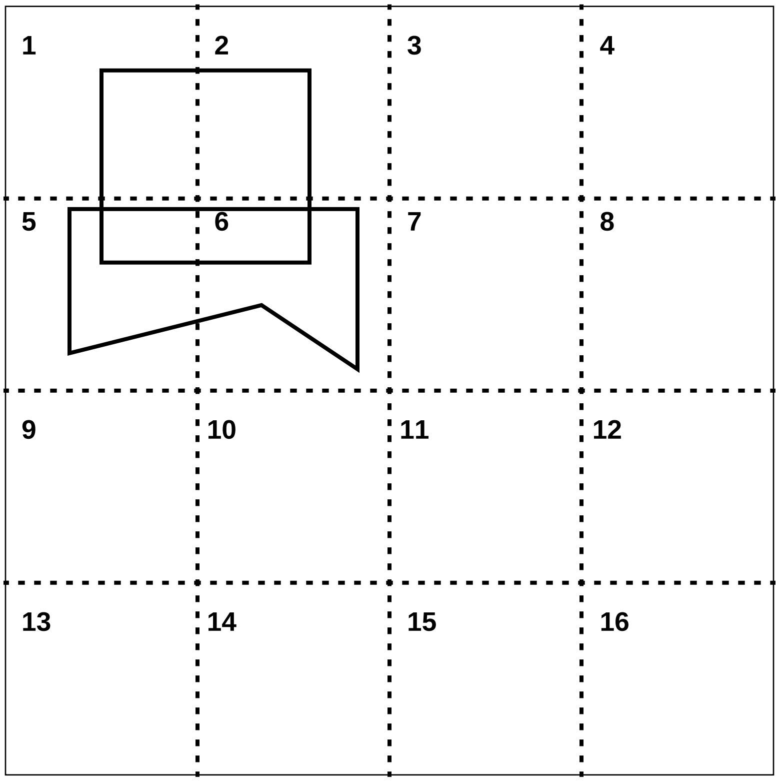
To reduce the number of times a complex predicate needs to be applied, variable-sized tile indexing uses a mechanism similar to spatial partitioning. To use this mechanism, select a tiling level, called the **groupcode level**, that results in tiles larger than any variable-sized tile generated for all the geometries in the layer or data set of interest. Each tile at the specified groupcode level can be considered a spatial partition. This reduces the size of the data set on which the complex predicate is evaluated. Example 1–2 illustrates this feature.

Example 1–2

```
SELECT r.sdo_gid
FROM layer_sdoindex r,
     window_sdoindex w
WHERE w.sdo_gid = 5
  AND r.sdo_group_code = w.sdo_groupcode
  AND (r.sdo_code BETWEEN w.sdo_code AND w.sdo_maxcode OR
       w.sdo_code BETWEEN r.sdo_code AND r.sdo_maxcode);
```

In Figure 1–7, consider the domain partitioned into 16 subregions. If a join compares tiles from the two objects, under normal circumstances the join operation would process tiles from the entire domain, searching for tiles that interact. However, if you constrain the processing to common partitions, then only partitions 5 and 6 would need to be processed. This may result in substantial performance improvements.

Figure 1–7 Spatially Partitioning Data



1.6 Partitioned Point Data

Spatial Cartridge has an enhanced spatial indexing mechanism capable of handling very large data sets consisting of complex geometries. For applications handling point data sets that are several tens of gigabytes or larger, further performance gains can be achieved by using Oracle8 table partitioning features.

Table partitioning is only available with the Partitioning Option of Oracle8 Enterprise Edition. If the Partitioning Option is available to you, the preferred method is to use Oracle8 table partitioning in conjunction with Spatial Cartridge spatial indexing (see the *Oracle8 Concepts* guide for a description of Oracle8 Partitioning). A technical white paper titled, *“Leveraging Oracle8 Partitioning and the Spatial Cartridge for*

Large Point Datasets,” describing the use of partitioning and spatial indexing for point data sets may be obtained from the Oracle corporate web site at:

```
http://www.oracle.com/st/cartridges/spatial/collateral
```

A previous release of Spatial Data Option (from which Spatial Cartridge has evolved) utilized its own version of table partitioning instead of spatial indexing. Chapter 4 briefly describes the old partitioning scheme for those customers with legacy point data sets. Any references to point data partitioning in this guide (such as the ”Partitioned Point Data Procedures” section in Chapter 5) refer to this legacy feature. While this feature is still available in Spatial Cartridge, the preferred approach is to use Oracle8 Partitioning Option and spatial indexing.

2
Loading Spatial Data

This chapter describes how to load spatial data into a database, including storing the data in a table and creating a spatial index for it.

2.1 Load Model

There are two steps involved in loading raw data into a spatial database such that it can be queried efficiently:

1. Loading the data into spatial tables
2. Creating or updating the index on the spatial tables

Table 2–1 through Table 2–4 show the format of the tables needed to store and index spatial data.

Table 2–1 <layername>_SDOLAYER

SDO_ORDCNT	SDO_LEVEL	SDO_NUMTILES	SDO_COORDSYS
<number>	<number>	<number>	<varchar>

Table 2–2 <layername>_SDODIM Table or View

SDO_DIMNUM	SDO_LB	SDO_UB	SDO_TOLERANCE	SDO_DIMNAME
<number>	<number>	<number>	<number>	<varchar>

Table 2–3 <layername>_SDOGEOM Table or View

SDO_GID	SDO_ESEQ	SDO_ETYPE	SDO_SEQ	SDO_X1	SDO_Y1	...	SDO_Xn	SDO_Yn
<number>	<number>	<number>	<number>	<number>	<number>	...	<number>	<number>

Table 2–4 <layername>_SDOINDEX Table

SDO_GID	SDO_CODE	SDO_MAXCODE	SDO_GROUPCODE	SDO_META
<number>	<raw>	<raw>	<raw>	<raw>

2.2 Load Process

The process of loading data can be classified into two categories:

- *Bulk loading of data*

  This process is used to load large volumes of data into the database and uses the SQL*Loader to load the data.

- *Transactional inserts*

  This process is used to insert relatively small amounts of data into the database and is analogous to the INSERT statement in SQL.

2.2.1 Bulk Loading

Bulk loading can be used to import large amounts of legacy or ASCII data into a spatial database. Bulk loading is accomplished using the SQL*Loader[1].

Example 2–1 shows the format of the raw data and control file that would be required to load the data into the SDOGEOM table with the layer name ROADS. You can choose any format of ASCII data as long you can write a SQL*Loader control file to load that data into the tables.

Assume that the ASCII data consists of a file with delimited columns, and separate rows fixed by the limits of the table with the following format.

Example 2–1

```
geometry rows:    GID, ESEQ, ETYPE, SEQ, LON1, LAT1, LON2, LAT2
```

[1] See the *Oracle Server Utilities User's Guide* for information on the SQL*Loader.

The coordinates in the geometry rows represent the end points of line segments, which taken together, represent a polygon. Example 2–2 shows the control file for loading the data into the geometry table.

Example 2–2

```
LOAD DATA INFILE *
INTO TABLE ROADS_SDOGEOM
FIELDS TERMINATED BY WHITESPACE TRAILING NULLCOLS
(SDO_GID INTEGER EXTERNAL,
SDO_ESEQ INTEGER EXTERNAL,
SDO_ETYPE INTEGER EXTERNAL,
SDO_SEQ INTEGER EXTERNAL,
SDO_X1 FLOAT EXTERNAL,
SDO_Y1 FLOAT EXTERNAL,
SDO_X2 FLOAT EXTERNAL,
SDO_Y2 FLOAT EXTERNAL)

BEGINDATA
1 0 3 0 -122.401200   37.805200 -122.401900   37.805200
1 0 3 1 -122.401900   37.805200 -122.402400   37.805500
1 0 3 2 -122.402400   37.805500 -122.403100   37.806000
1 0 3 3 -122.403100   37.806000 -122.404400   37.806800
1 0 3 4 -122.404400   37.806800 -122.401200   37.805200
1 1 3 0 -122.405900   37.806600 -122.407549   37.806394
1 1 3 1 -122.407549   37.806394 -122.408300   37.806300
1 1 3 2 -122.408300   37.806300 -122.409100   37.806200
1 1 3 3 -122.409100   37.806200 -122.405900   37.806600
2 0 2 0 -122.410800   37.806000 -122.412300   37.805800
2 0 2 1 -122.412300   37.805800 -122.414100   37.805600
2 0 2 2 -122.414100   37.805600 -122.412300   37.805800
2 0 2 3 -122.412300   37.805800 -122.410800   37.806000
3 0 1 0 -122.567474   38.643564
3 0 1 1 -126.345345   39.345345
```

Note that table ROADS_SDOGEOM exists in the schema before attempting the load.

In Example 2–3, the data resides in a single flat file and the data set consists of point, line string, and polygon data. The data uses fixed-position columns and overloaded table rows.

Example 2–3

```
SDO_GID  SDO_ESEQ  SDO_ETYPE  SDO_SEQ  SDO_X1  SDO_Y1  SDO_X2  SDO_Y2
```

The corresponding control file for this format of input data would be:

```
LOAD DATA INFILE *
INTO TABLE NEW_SDOGEOM
(SDO_GID POSITION (1:5) INTEGER EXTERNAL,
SDO_ESEQ POSITION (7:10) INTEGER EXTERNAL,
SDO_ETYPE POSITION (12:15) INTEGER EXTERNAL,
SDO_SEQ POSITION (17:21) INTEGER EXTERNAL,
SDO_X1 POSITION (23:35) FLOAT EXTERNAL,
SDO_Y1 POSITION (37:48) FLOAT EXTERNAL,
SDO_X2 POSITION (50:62) FLOAT EXTERNAL,
SDO_Y2 POSITION (64:75) FLOAT EXTERNAL)

BEGINDATA
1     0    3    0    -122.401200    37.805200   -122.401900    37.805200
1     0    3    1    -122.401900    37.805200   -122.402400    37.805500
1     0    3    2    -122.402400    37.805500   -122.403100    37.806000
1     0    3    3    -122.403100    37.806000   -122.404400    37.806800
1     0    3    4    -122.404400    37.806800   -122.401200    37.805200
1     1    3    0    -122.405900    37.806600   -122.407549    37.806394
1     1    3    1    -122.407549    37.806394   -122.408300    37.806300
1     1    3    2    -122.408300    37.806300   -122.409100    37.806200
1     1    3    3    -122.409100    37.806200   -122.405900    37.806600
2     0    2    0    -122.410800    37.806000   -122.412300    37.805800
2     0    2    1    -122.412300    37.805800   -122.414100    37.805600
2     0    2    2    -122.414100    37.805600   -122.412300    37.805800
2     0    2    3    -122.412300    37.805800   -122.410800    37.806000
3     0    1    0    -122.567474    38.643564
3     0    1    1    -126.345345    39.345345
```

2.2.2 Transactional Insert Using SQL

Spatial Cartridge uses standard Oracle8 tables that can be accessed or loaded with standard SQL syntax. Example 2–4 loads data for a geometry (GID 17) consisting of a polygon with four sides that contains both a hole and point. Notice that the first coordinate of the polygon (5, 20) is repeated at the end to close the polygon.

Example 2–4

```
INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                              SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
  VALUES (17, 0, 3, 0, 5, 20, 5, 30, 10, 30, 10, 20, 5, 20);
```

```
   -- hole
INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                              SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
  VALUES (17, 1, 3, 0, 8, 21, 8, 24, 9, 24, 9, 21, 8, 21);

   -- point
INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1)
  VALUES (17, 2, 1, 0, 9, 29);
```

The SQL INSERT statement inserts one row of data per call. In Example 2–4, the table had enough columns to store the polygon in a single row. However, if your table had fewer columns (or your polygon had more points), you would have to perform mulitple inserts in order to match the table structure; the data would not wrap automatically to the next row. To load a large geometry, repeat the SDO_GID, SDO_ESEQ, and SDO_ETYPE, and increment the SDO_SEQ for each line as shown in Example 2–5.

Example 2–5

```
INSERT INTO SAMPLE2_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                              SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
  VALUES (18, 0, 3, 0, 1, 15, 1, 16, 2, 17, 3, 17, 4, 18);

INSERT INTO SAMPLE2_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                              SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
  VALUES (18, 0, 3, 1, 4, 18, 5, 18, 6, 19, 7, 18, 6, 17);

INSERT INTO SAMPLE2_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                              SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
  VALUES (18, 0, 3, 2, 6, 17, 7, 16, 7, 15, 6, 14, 7, 13);

INSERT INTO SAMPLE2_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                              SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
  VALUES (18, 0, 3, 3, 7, 13, 6, 12, 5, 13, 4, 13, 3, 14);

INSERT INTO SAMPLE2_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                              SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
SDO_Y3)
```

```
        VALUES (18, 0, 3, 4, 3, 14, 2, 14, 1, 15);
```

2.2.3 Transactional Insert Using Spatial Geometry Functions

Spatial Cartridge provides two functions to facilitate inserting data into spatial tables. A benefit to using these functions is that the issue of row-wrapping when loading elements with multiple points is handled automatically by these functions.

There are two steps to incrementally add data to the spatial tables:

1. Initialize the element that needs to be stored. Note that this process does not fill in any coordinate information for the element. Two parameters are passed to the `SDO_GEOM.INIT_ELEMENT()` function, which initializes the element:

   - Name of the layer (for example, ROADS)
   - GID that is a unique identifier for the geometry

   The `SDO_GEOM.INIT_ELEMENT()` function returns the sequence number of the element in the geometry. This sequence number is required as a parameter to the `SDO_GEOM.ADD_NODES()` procedure.

2. Fill in the coordinate information for the element using the `SDO_GEOM.ADD_NODES()` procedure. This procedure takes the following parameters:

   - Name of the layer
   - GID
   - Sequence number of the element
   - Element type
   - List of vertices in the geometry, specified as a series of X,Y coordinate pairs

     Note that you must explicitly close a polygon by repeating the coordinates of the first vertex as the last vertex.

In Example 2–6, a simple polygon, geometry number 1234, consisting of five vertices needs to be stored. The first step is to call `SDO_GEOM.INIT_ELEMENT()` to initialize the element.

Example 2–6

```
elem_value := sdo_geom.init_element('ROADS', 1234);
```

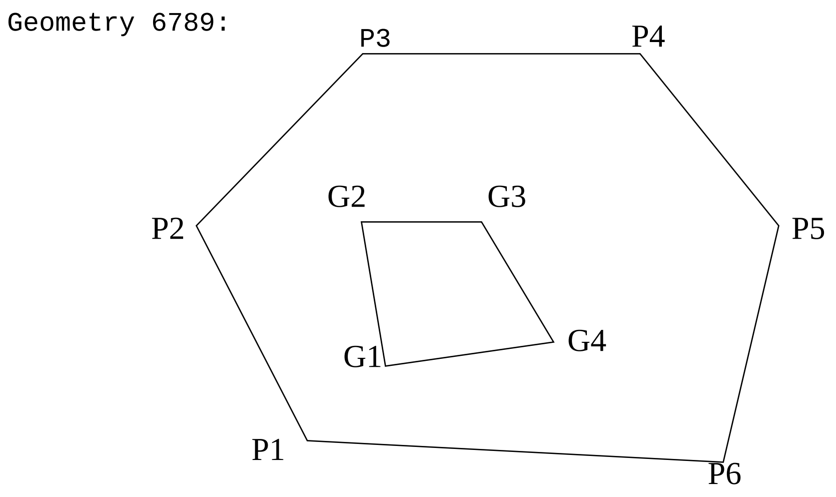
Next, call `SDO_GEOM.ADD_NODES()` to fill in the attributes of the polygon. The vertices can be added in either clockwise or counter-clockwise order.

```
sdo_geom.add_nodes('ROADS', 1234, elem_value, sdo_geom.polygon_type, Ax,
Ay, Bx, By,  Cx, Cy,  Dx, Dy,  Ex, Ey,  Ax, Ay);
```

Close the polygon by repeating the first vertex (A_x,A_y) as the last vertex.

In Example 2–7, assume that the geometry shown in Figure 2–1 needs to be stored. The geometry consists of a polygon with a hole in it. Note that both calls to the `SDO_GEOM.ADD_NODES()` procedure are made with the same GID (6789) because this is a single object even though it is composed of two elements.

Figure 2–1 Polygon with a Hole



Example 2–7

```
val1 := sdo_geom.init_element('PARKS', 6789);
sdo_geom.add_nodes('PARKS', 6789, val1,  SDO_GEOM.POLYGON_TYPE,  $P1_x$, $P1_y$,
$P2_x$, $P2_y$,  $P3_x$, $P3_y$,  $P4_x$, $P4_y$,  $P5_x$, $P5_y$,  $P6_x$, $P6_y$,  $P1_x$, $P1_y$);
val2 := sdo_geom.init_element('PARKS', 6789);
sdo_geom.add_nodes('PARKS', 6789, val2, SDO_GEOM.POLYGON_TYPE, $G1_x$, $G1_y$,
$G2_x$, $G2_y$,  $G3_x$, $G3_y$,  $G4_x$, $G4_y$,  $G1_x$, $G1_y$);
```

2.3 Index Creation

Once data has been loaded into the spatial tables through either bulk or transactional loading, a spatial index needs to be created on the tables for efficient access to the data.

Create an Oracle8 table called <layername>_SDOINDEX as follows:

```
SQL>  create table <layername>_SDOINDEX
   2  (
   3    SDO_GID integer,
   4    SDO_CODE raw(255)
   5    );
```

For a bulk load, you can call the `SDO_ADMIN.POPULATE_INDEX()` procedure once to tessellate the geometry table and add the generated tiles to the spatial index table. The argument to this procedure is simply the name of the layer. The level to which the geometry should be tessellated, and whether to use the fixed-size or variable-sized tile indexing technique is determined by values in the <layername>_SDOLAYER table.

If data is updated in or deleted from a specific geometry table, you can call `SDO_ADMIN.UPDATE_INDEX()` to update the index for one SDO_GID. The arguments to this procedure are the name of the layer and the SDO_GID of the designated geometry.

See Chapter 5, "Administrative Procedures" for a complete description of the `SDO_ADMIN.POPULATE_INDEX()` and `SDO_ADMIN.UPDATE_INDEX()` procedures.

2.3.1 Choosing a Tessellation Algorithm

Spatial Cartridge provides two methods for spatial indexing. Fixed-size tiling is recommended for all production applications. For advanced development applications, you may want to experiment with variable-sized tiling, which theoretically could provide better selectivity in some data sets.

Which tessellation algorithm is used by the `SDO_ADMIN.POPULATE_INDEX()` and `SDO_ADMIN.UPDATE_INDEX()` procedures is determined by the values of the SDO_LAYER and SDO_NUMTILES columns in the <layername>_SDOLAYER table as follows:

SDO_LEVEL	SDO_NUMTILES	Action
NULL	NULL	Error

SDO_LEVEL	SDO_NUMTILES	Action
>= 1	NULL	Fixed-size tiling
>= 1	>= 1	Indexing with variable-sized tiles. The SDO_LEVEL column defines the partition bucket size. The SDO_NUMTILES column defines the number of tiles to generate per geometry. Note: variable-sized tiling is for experimentation purposes only.
NULL	>= 1	Not supported

2.3.2 Spatial Indexing with Fixed-Size Tiles

Oracle recommends using fixed-size cover tiles for indexing a geometry.

The fixed-size tile algorithm is expressed as a level referring to the number of tessellations performed. To use fixed-size tile indexing, set the SDO_NUMTILES column in the <layername>_SDOLAYER table to NULL and the SDO_LEVEL column to the desired tiling level. The relationship between the tiling level and the resulting size of the tiles is dependent on the domain of the layer.

The domain used for indexing is defined by the upper and lower boundaries of each dimension stored in the <layername>_SDODIM table. A typical domain in a GIS application could be -90 to 90 degrees for latitude, and -180 to 180 degrees for longitude[1], as represented in Figure 2–2.

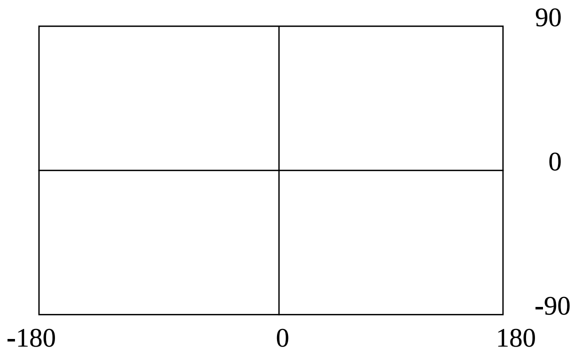
[1] The transference of the domain onto a sphere or Mercator projection is left to GIS (or other) application programmers. Spatial Cartridge treats the domain as a conventional X by Y rectangle.

Figure 2–2 Sample GIS Domain

90
-90
-180
180

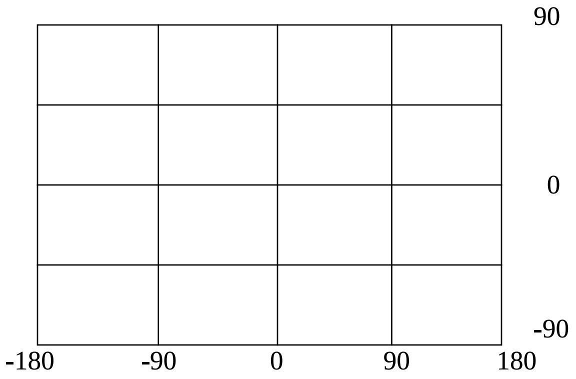
If the SDO_LEVEL column is set to 1, then the tiles created by the indexing mechanism are the same size as tiles at the first level of tessellation. Each tile would be 180 degrees by 90 degrees as shown in Figure 2–3.

Figure 2–3 Fixed-Size Tiling at Level 1



The formula for the number of fixed-size tiles is 4^n where n is the number of tessellations, stored in the SDO_LEVEL column. Figure 2–4 shows fixed-size tiling at level 2. In this figure, each tile is 90 degrees by 45 degrees.

Figure 2–4 Fixed-Size Tiling at Level 2



The size of a tile can be determined by applying the following formula to each dimension:

length = (upper_bound - lower_bound) / 2 ^ sdo_level

The length refers to the length of the tile along the specified dimension. Applying this formula to the tiling shown in Figure 2–4 yields the following sizes:

```
length for dimension X = (180 - (-180) ) / 2^2
                       = (360)  / 4
                       = 90
length for dimension Y = (90 - (-90) ) / 2^2
                       = (180) / 4
                       = 45
```

Thus, at level 2 the tiles are 90x45 degrees in size. As the number of levels increases, the tiles become smaller and smaller. Smaller tiles provide a more precise fit of the tiles over the geometry being indexed. However, because the number of tiles generated is unbounded, you must take into account the performance implications of using higher levels. The `SDO_TUNE.ESTIMATE_TILING_LEVEL()` function can be used to determine an appropriate level for indexing with fixed-size tiles. See Chapter 6 for a description of this procedure.

Besides the performance aspects related to selecting a fixed-size tile, tessellating the geometry into fixed-size tiles might have benefits related to the type of data being stored, such as using tiles sized to represent 1-acre farm plots, city blocks, or individual pixels on a display. Data modeling is an important part any database design, and is essential in a spatial database where the data often represents actual physical locations.

In Example 2–8, assume that data has been loaded into a layer called ROADS, and you want to create a spatial index on that data.

Example 2–8

To create a spatial index, create a table ROADS_SDOINDEX and invoke the following procedure:

```
sdo_admin.populate_index('ROADS');
```

The value in the SDO_LEVEL column of the ROADS_SDOLAYER table can be used as a tuning parameter while tessellating objects. Increasing the level increases the number of tiles to provide a more precise fit of the tiles over the object. See the description of the `ESTIMATE_TILING_LEVEL()`function in Chapter 6 for information on estimating the tiling level in several different ways.

After the `SDO_ADMIN.POPULATE_INDEX()` procedure has been called to fill the spatial index, you should also create a concatenated index using the SDO_CODE and SDO_GID columns. The concatenated index helps the join to the <layername>_SDOGEOM table during a query. The SDO_GID values from the primary filter will come from the index instead of from the table.

If a geometry with an SDO_GID 5944 has been added to the spatial tables, update the index with the following procedure:

```
sdo_admin.update_index('ROADS', 5944);
```

The `SDO_ADMIN.POPULATE_INDEX()`and `SDO_ADMIN.UPDATE_INDEX()`procedures behave differently from the CREATE INDEX statement in SQL. An implicit commit is not executed after the procedures are called. Therefore these transactions can be rolled back.

The `SDO_ADMIN.POPULATE_INDEX()` procedure operates as a single transaction. To reduce the amount of rollback required to execute this procedure, you can write a routine that loops and calls `SDO_ADMIN.UPDATE_INDEX()` . See Section A.1.1.1, “cr_spatial_index.sql Script” for more information.

2.3.3 Spatial Indexing with Variable-Sized Tiles

Spatial indexing with variable-sized tiles is not recommended for production systems. Variable-sized tiling is included in Spatial Cartridge primarily for experimentation purposes.

To use variable-sized tiling, the SDO_LEVEL and SDO_NUMTILES columns must be set in the <layername>_SDOLAYER table.

The SDO_NUMTILES column determines the number of tiles that will be used to cover a geometry being indexed. Typically this value is small, such as 4 or 8 tiles. However, the larger the number of tiles, the better the tiles will fit the geometry being covered. This increases the selectivity of the primary filter. See Section 3.3.2 and Section 3.3.3 for a discussion of primary and secondary filters.

The SDO_LEVEL column indicates the spatial partitioning level for the generated tiles. See Section 1.5.3 for a description of the spatial partitioning utilized by Spatial Cartridge when using variable-sized tiles.

Setting the proper SDO_LEVEL value is more art than science. One approach would be use the `SDO_TUNE.ESTIMATE_TILING_LEVEL()` function to determine an appropriate starting SDO_LEVEL value, and then compare the performance with slightly higher or lower values.

3

Querying Spatial Data

This chapter describes how the structures of a Spatial Cartridge layer are used to resolve spatial queries and spatial joins. For the sake of clarity, the examples all use fixed-size tiling.

3.1 Query Model

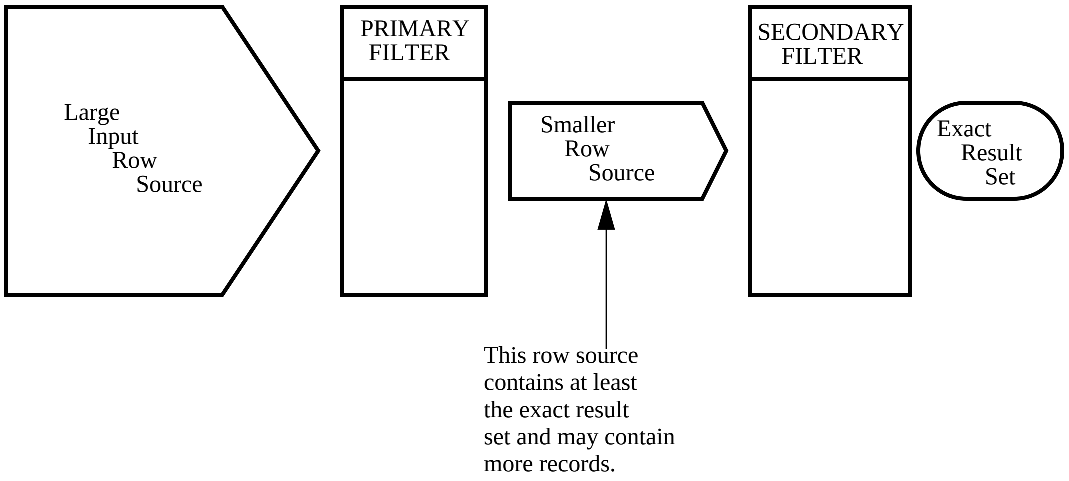
Spatial Cartridge uses a *two-tier* query model to resolve spatial queries and spatial joins. The term is used to indicate that two distinct operations are performed in order to resolve queries. The output of both operations yields the exact result set.

The two operations are referred to as *primary* and *secondary* filter operations.

- The primary filter permits fast selection of a small number of candidate records to pass along to the secondary filter. The primary filter uses approximations in order to reduce computational complexity and is considered a lower cost filter.
- The secondary filter applies exact computational geometry to the result set of the primary filter. These exact computations yield the final answer to a query. The secondary filter operations are computationally more expensive, but they are only applied to the relatively small result set from the primary filter.

Figure 3–1 illustrates the relationship between the primary and secondary filters.

Figure 3–1 Query Model



Spatial Cartridge uses a spatial index to implement the primary filter. This is described in detail in following sections.

A function used as a secondary filter is `SDO_GEOM.RELATE()`, which determines the spatial relationship between two given geometries, such as whether they touch, overlap, or if one is inside the other.

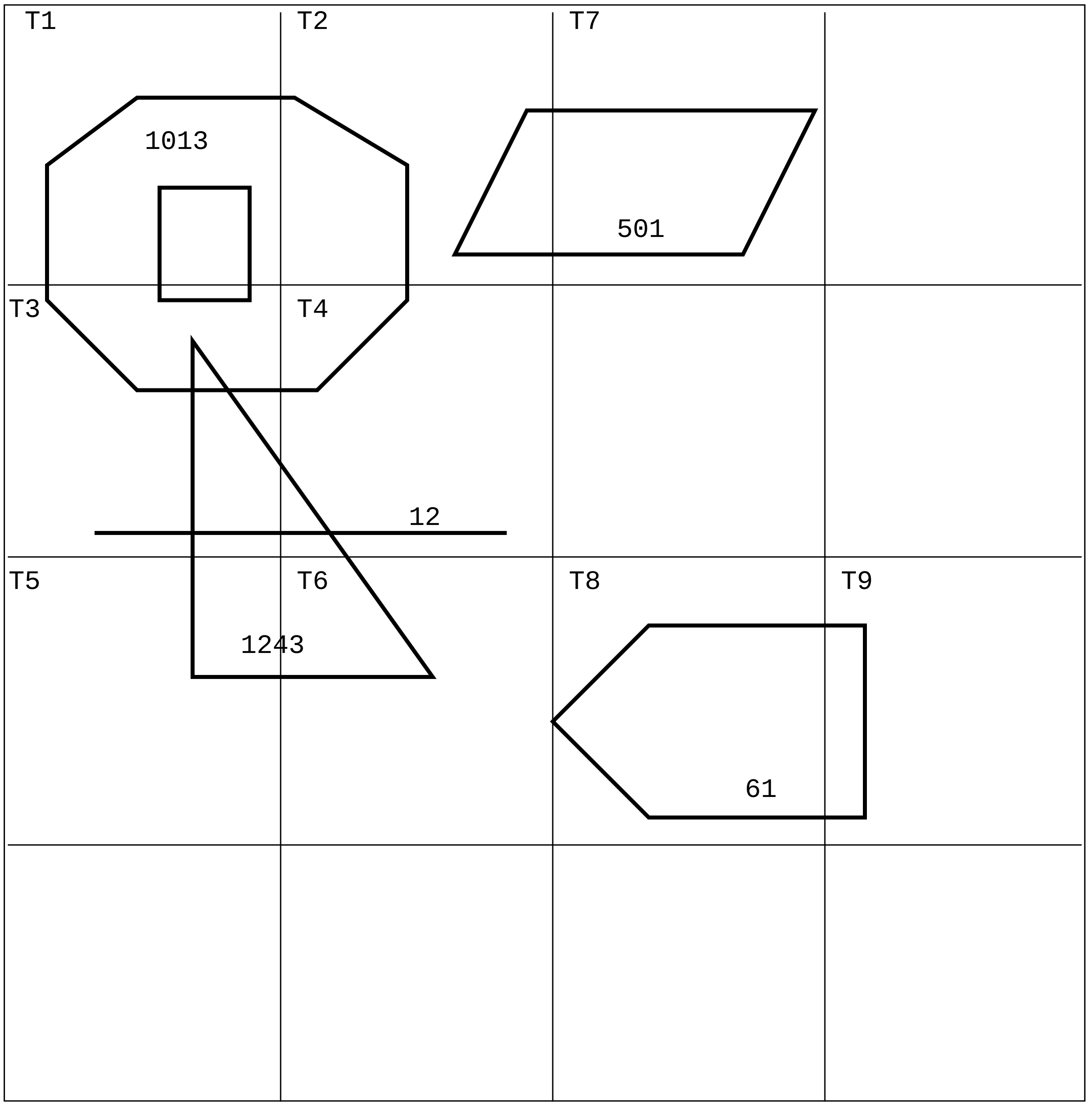
Spatial Cartridge does not require the use of both the primary and secondary filters. In some cases, just using the primary filter is sufficient. For example, a *zoom* feature in a mapping application queries for data that overlaps a rectangle representing visible boundaries. The primary filter very quickly returns a superset of the query. The mapping application can then apply clipping routines to display the target area.

3.2 Spatial Data Model

An important concept in the spatial data model is that each element is represented in the <layername>_SDOINDEX table by a set of exclusive and exhaustive tiles. This means that the tiles fully cover the object (**exhaustive**) and that no tiles overlap each other (**exclusive**).

Consider the following layer containing several objects in Figure 3–2. Each object is labeled with its SDO_GID. The relevant tiles are labeled with ‘Tn’.

Figure 3–2 Tessellated Layer with Multiple Objects



The Spatial Cartridge layer tables would have the following information stored in them for these geometries as shown in Table 3–1, Table 3–2, and Table 3–3.

Table 3–1 <layername>_SDOLAYER

SDO_ORDCNT (number)	SDO_LEVEL (number)	SDO_NUMTILES (number)
4	2	NULL

Table 3–2 <layername>_SDOGEOM

SDO_GID (number)	SDO_ESEQ (number)	SDO_ETYPE (number)	SDO_SEQ (number)	SDO_X1 (number)	SDO_Y1 (number)	SDO_X2 (number)	SDO_Y2 (number)
1013	0	3	0	P1(X)	P1(Y)	P2(X)	P2(Y)
1013	0	3	1	P2(X)	P2(Y)	P3(X)	P3(Y)
1013	0	3	2	P3(X)	P3(Y)	P4(X)	P4(Y)
1013	0	3	3	P4(X)	P4(Y)	P5(X)	P5(Y)
1013	0	3	4	P5(X)	P5(Y)	P6(X)	P6(Y)
1013	0	3	5	P6(X)	P6(Y)	P7(X)	P7(Y)
1013	0	3	6	P7(X)	P7(Y)	P8(X)	P8(Y)
1013	0	3	7	P8(X)	P8(Y)	P1(X)	P1(Y)
1013	1	3	0	G1(X)	G1(Y)	G2(X)	G2(Y)
1013	1	3	1	G2(X)	G2(Y)	G3(X)	G3(Y)
1013	1	3	2	G3(X)	G3(Y)	G4(X)	G4(Y)
1013	1	3	3	G4(X)	G4(Y)	G1(X)	G1(Y)
501	0	3	0	A1(X)	A1(Y)	A2(X)	A2(Y)
501	0	3	1	A2(X)	A2(Y)	A3(X)	A3(Y)
501	0	3	2	A3(X)	A3(Y)	A4(X)	A4(Y)
501	0	3	3	A4(X)	A4(Y)	A1(X)	A1(Y)
1243	0	3	0	B1(X)	B1(Y)	B2(X)	B2(Y)
1243	0	3	1	B2(X)	B2(Y)	B3(X)	B3(Y)
1243	0	3	2	B3(X)	B3(Y)	B1(X)	B1(Y)
12	0	2	0	D1(X)	D1(Y)	D2(X)	D2(Y)
61	0	3	0	C1(X)	C1(Y)	C2(X)	C2(Y)
61	0	3	1	C2(X)	C2(Y)	C3(X)	C3(Y)
61	0	3	2	C3(X)	C3(Y)	C4(X)	C4(Y)
61	0	3	3	C4(X)	C4(Y)	C5(X)	C5(Y)
61	0	3	4	C5(X)	C5(Y)	C1(X)	C1(Y)

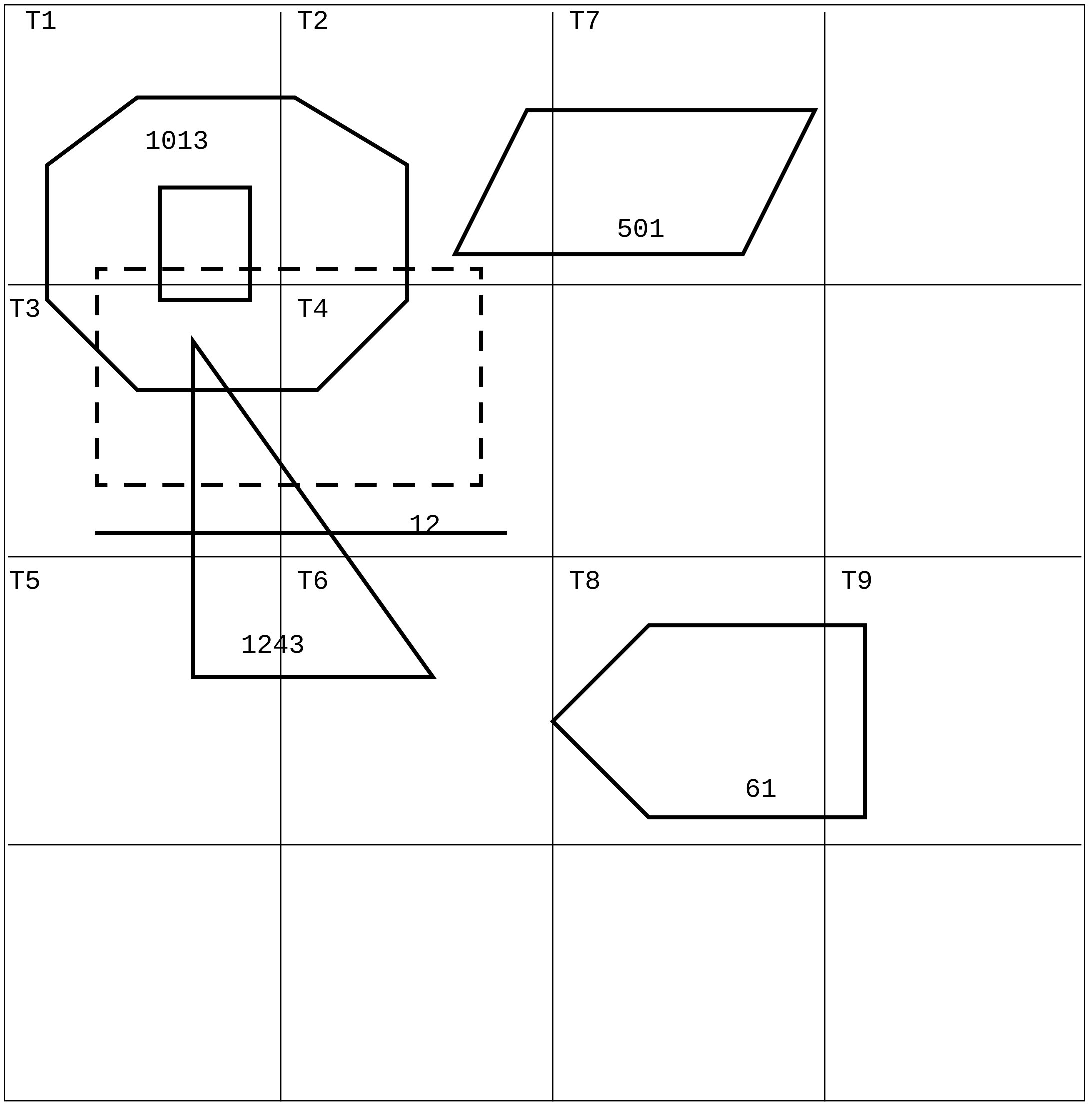
Table 3–3 <layername>_SDOINDEX

SDO_GID (number)	SDO_CODE (raw)
1013	T1
1013	T2
1013	T3
1013	T4
501	T2
501	T7
1243	T3
1243	T4
1243	T5
1243	T6
12	T3
12	T4
61	T8
61	T9

3.3 Spatial Query

A typical spatial query is to request all objects that lie within a defined fence or window. A **query window** is shown in Figure 3–3 by the dotted line box. A dynamic query window refers to a fence that is not defined in the database, but that must be defined and indexed prior to using it.

Figure 3–3 Tessellated Layer with a Query Window



3.3.1 Dynamic Query Window

If a query window does not already exist in the database, you must first insert it and create an index for it. Because not all Oracle users necessarily have insert privileges, Spatial Cartridge includes the SDO_WINDOW PL*SQL package. See Chapter 8, “Window Functions” for more information.

The SDO_WINDOW package is not automatically installed when you install Spatial Cartridge. This allows a DBA to control the schema under which this package operates. Choose an Oracle user who has insert privilege and compile the SDO_WINDOW package under that user. For example, you could choose the mdsys Oracle user:

```
sqlplus mdsys/password
SQL> @$ORACLE_HOME/md/admin/sdowin.sql
SQL> @$ORACLE_HOME/md/admin/prvtwin.plb
```

After compiling, the routines are available for use. When you call a routine in this package, and the routine performs an INSERT operation, the insert will occur under the mdsys schema. Note that it is not a requirement to use the mdsys account. You can select any Oracle user with insert privileges.

If you need to perform other INSERT, UPDATE, or DELETE operations, and you cannot guarantee that the user of your application has those privileges, you can write your own PL*SQL package similar to the SDO_WINDOW package. You will have to compile your package under a user with the required database privileges.

3.3.2 Primary Filter

To resolve the window query shown in Figure 3–3, build a layer for the query fence if it is not already defined:

```
SQL> EXECUTE MDSYS.SDO_WINDOW.CREATE_WINDOW_LAYER (fencelayer, DIMNUM1, LB1,
UB1, TOLERANCE1, DIMNAME1, DIMNUM2, LB2, UB2, TOLERANCE2, DIMNAME2);
```

Next, insert the ordinates for the query fence into the layer tables:

```
SQL> EXECUTE DBMS_OUTPUT.PUTLINE(TO_CHAR(MDSYS.SDO_WINDOW.BUILD_WINDOW_FIXED
(comp_user, fencelayer, SDO_ETYPE, TILE_SIZE, X1, Y1, X2, Y2, X3, Y3, X4, Y4,
X1, Y1)));
```

Query SDO_LEVEL from the <fencelayer>_SDOLAYER table to pass the correct TILE_SIZE to the `SDO_WINDOW.BUILD_WINDOW_FIXED()` procedure.

Now you can construct a query that joins the index of the query window to the appropriate layer index and determines all elements that have these tiles in common. The following SQL query form is used:

```
SELECT DISTINCT A.SDO_GID
FROM <layer1>_SDOINDEX A, <fencelayer>_SDOINDEX B
WHERE A.SDO_CODE = B.SDO_CODE
  AND B.SDO_GID = {GID returned from SDO_WINDOW.BUILD_WINDOW_FIXED};
```

The result set of this query is the primary filter set. In this case, the result set is:

```
 { 1013,501,1243,12 }
```

3.3.3 Secondary Filter

The secondary filter performs exact geometry calculations of the tiles selected by the primary filter. The following example shows the primary and secondary filters:

```
SELECT SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3, SDO_Y3, SDO_X4, SDO_Y4
FROM <layer1>_SDOGEOM,
(
SELECT SDO_GID GID1
   FROM (
          SELECT DISTINCT A.SDO_GID
             FROM <layer1>_SDOINDEX A,
                  <fencelayer>_SDOINDEX B
             WHERE A.SDO_CODE = B.SDO_CODE
              AND B.SDO_GID = {GID returned from SDO_WINDOW.BUILD_WINDOW_FIXED}

         )
   WHERE SDO_GEOM.RELATE('<layer1>', GID1, 'ANYINTERACT', '<fence>', 1) = 'TRUE'
   )
WHERE SDO_GID = GID1;
```

This query would return all the geometry IDs that lie within or overlap the window. In this example, the results of the secondary filter would be:

```
{1243,1013}
```

The example in this section uses the `SDO_GEOM.RELATE()` secondary filter. Both the INTERACT() and RELATE() secondary filters are overloaded functions. For better performance, use the versions that explicitly list the coordinates of the query window whenever possible. See Chapter 7, “Geometry Functions” for details on using these functions.

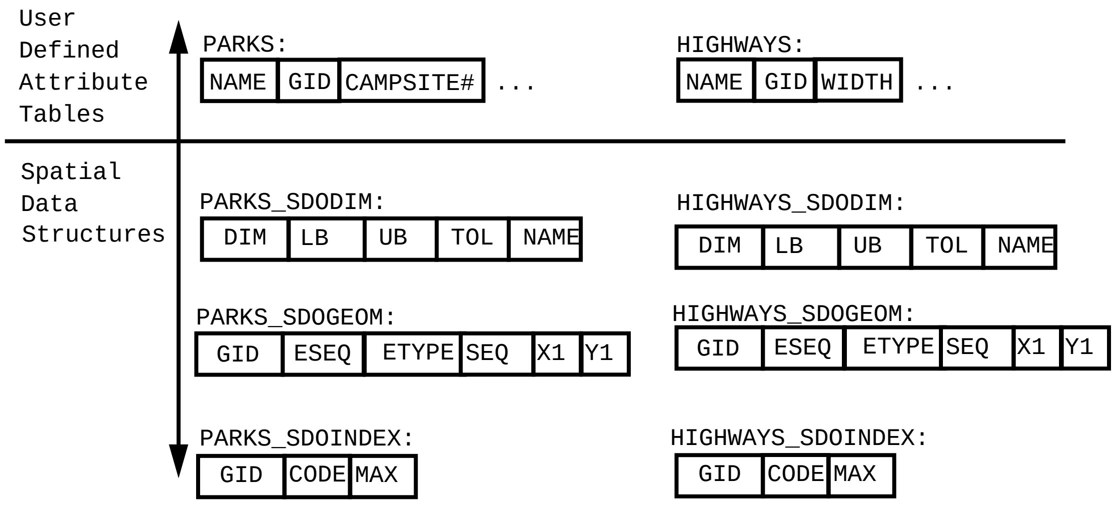
3.4 Spatial Join

A spatial join is the same as a regular join except that the predicate involves a spatial operator. In Spatial Cartridge, a spatial join takes place between two layers; specifically, two <layername>_SDOINDEX tables are joined.

Spatial joins can be used to answer questions such as, "which highways cross national parks?"

This query could be resolved by joining a layer that stores national park geometries with one that stores highway geometries.Figure 3–4 illustrates how the join would be accomplished for this example.

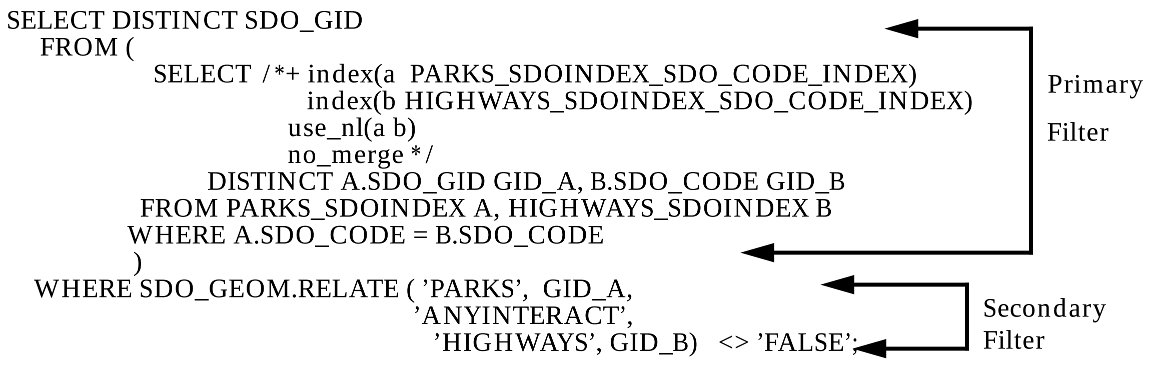
Figure 3–4 Spatial Join of Two Layers



The PRIMARY filter would identify pairs of park GIDs and highway GIDs that cross in the index. The query that performs the primary filter join (assuming fixed-size tile indexing) is as follows:

```
SELECT DISTINCT A.SDO_GID,B.SDO_GID
     FROM PARKS_SDOINDEX A, HIGHWAYS_SDOINDEX B
     WHERE A.SDO_CODE = B.SDO_CODE
```

The result set of the primary filter must be passed through the secondary filter to get the exact set of parks/highways GID pairs that cross. The full query is shown in the following example:



```
SELECT DISTINCT SDO_GID
    FROM (
              SELECT  /*+ index(a  PARKS_SDOINDEX_SDO_CODE_INDEX)
                          index(b HIGHWAYS_SDOINDEX_SDO_CODE_INDEX)
                         use_nl(a b)
                         no_merge */
                   DISTINCT A.SDO_GID GID_A, B.SDO_CODE GID_B
             FROM PARKS_SDOINDEX A, HIGHWAYS_SDOINDEX B
            WHERE A.SDO_CODE = B.SDO_CODE
            )
   WHERE SDO_GEOM.RELATE ( 'PARKS',  GID_A,
                                       'ANYINTERACT',
                                         'HIGHWAYS', GID_B)   <> 'FALSE';
```

Suppose the original query had asked, "which *4-lane* highways cross national parks?" You could modify the preceding SQL statement to join back to the HIGHWAYS table where HIGHWAYS.WIDTH=4. This combination of spatial and relational attributes in a single query is one of the essential reasons for using Spatial Cartridge.

4 Partitioning Point Data

Spatial Cartridge provides the essential functions, procedures, and scripts for using and managing both spatially indexed data and partitioned point data. The information in this chapter is relevant only to users utilizing table partitioning for very large quantities of point data.

4.1 Overview

Partitioning is a technique where data is loaded into tables that automatically subdivide when a predefined maximum size is reached. During subdivision, data is moved from the parent partition to the child partitions and the parent partition is dropped. Storage parameters for child partitions are inherited from the root partition and can be changed at any time.

A partitioned table has a partition key that is an HHCODE column created by encoding multidimensional point data using the `SDO_ENCODE()` function. In the partitioning process, at each subdivision, data is subdivided into 2^n partitions where *n* is the number of dimensions encoded in the HHCODE column. You can encode up to 32 dimensions using Spatial Cartridge.

4.2 Partitioning Process

This guide does not attempt to provide the information necessary for fully utilizing table partitioning for point data. The following is a high-level description of the partitioning process:

1. Start with an Oracle8 table containing multidimensional point data. For example, columns of X and Y coordinate data from a blueprint or map.
2. Create a table or view from the original Oracle8 table containing the columns you want, plus a new HHCODE column.

An HHCODE column is a new data type used to encode multiple dimensions into a unique orderable value. HHCODE is not a point, but rather a bounded cell representing an object space in as many dimensions as have been defined. An HHCODE data type is defined as RAW(255).

3. Create the HHCODE data type by encoding multiple dimensions into a single value using the `SDO_ENCODE()` function. The HHCODE data type will be used as the partition key.
4. Register a partitioned table in the Spatial Cartridge data dictionary using the `SDO_ADMIN.REGISTER_PARTITION_INFO()` procedure. This procedure takes the name of a table, the name of the partition key column, and the maximum number of records you want stored in a partition before it subdivides.
5. Call the `SDO_ADMIN.PARTITION()` procedure with the name of the table or view containing the partition key column and the tablespace in which the partitions should be created. In this step, the data is partitioned based on dimensions encoded in the HHCODE column.
6. If the underlying table has constraints, grants, or triggers, the owner needs to use the `SDO_ADMIN.PROPAGATE_GRANTS()` procedure to set those properties on the partitions.
7. To add more partitioned point data, load the data into a table, and call `SDO_ADMIN.PARTITION()` again. The dimensions encoded in the HHCODE column must have the same boundaries to be loaded into the existing partitioned table.
8. After you have added data multiple times, or after adding or deleting a large amount of data, there may be partitions that exceed the high-water mark or there may be partitions that can be merged. Call the `SDO_ADMIN.REPARTITION()` procedure to reorganize the partitioned table. Repartitioning is a computation-intensive task that should be performed only when necessary.

4.3 Function Details

See the following for details of the functions supporting partitioned point data:

- Chapter 5, "Administrative Procedures"
  - SDO_ADMIN.ALTER_HIGH_WATER_MARK
  - SDO_ADMIN.DROP_PARTITION_INFO
  - SDO_ADMIN.PARTITION
  - SDO_ADMIN.PROPAGATE_GRANTS

  - SDO_ADMIN.REGISTER_PARTITION
  - SDO_ADMIN.REPARTITION
  - SDO_ADMIN.VERIFY_PARTITIONS
- Chapter 9, “Partitioned Point Data Functions”
  - SDO_BVALUETODIM
  - SDO_COMPARE
  - SDO_DATETODIM
  - SDO_DECODE
  - SDO_ENCODE
  - SDO_TO_BVALUE
  - SDO_TO_DATE
- Appendix A, “Sample SQL Scripts and Tuning Tips”
  - altpart.sql
  - drppart.sql
  - sdogrant.sql

5 Administrative Procedures

The SDO_ADMIN procedures create and maintain spatial structures in the database, and are used to perform the following tasks:

- Tessellate entries in a geometry table and place them in a spatial index table
- Register and manipulate partitioned spatial tables (partitioned tables are used only for large volumes of point data)
- Verify spatial index and partitioned spatial table information

This chapter contains descriptions of the administrative procedures used for working with either spatially indexed geometric data or partitioned point data. These data structures are mutually exclusive and the procedures only work with the structure for which they are designed.

Table 5–1 lists the administrative procedures for working with spatially indexed geometry-based data. Table 5–2 later in this chapter lists procedures for working with partitioned point data.

Table 5–1 Administrative Procedures for Spatially Indexed Data

Procedure	Data Structure	Description
SDO_ADMIN.POPULATE_INDEX	Geometric objects	Generates a spatial index for the geometry table using either a set number of tiles, or a fixed-size tile.
SDO_ADMIN.SDO_CODE_SIZE	Geometric objects	Determines the required sizes for SDO_CODE and SDO_MAXCODE.
SDO_ADMIN.UPDATE_INDEX	Geometric objects	Updates the spatial index based on changes to the geometry table.
SDO_ADMIN.VERIFY_LAYER	Geometric objects	Checks for the existence of geometry and spatial index tables.

SDO_ADMIN.POPULATE_INDEX

Purpose

This procedure tessellates a list of geometric objects created by selecting all the entries in the geometry table that do not have corresponding entries in the spatial index table.

This procedure can generate either fixed or variable-sized tiles depending on values stored in the <layername>_SDOLAYER table.

Syntax

SDO_ADMIN.POPULATE_INDEX (*layername*)

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the names of the geometry and spatial index tables. Data type is VARCHAR2.

Usage Notes

Consider the following when using this procedure:

- The <layername>_SDOINDEX table must be created prior to calling this procedure. Use the SQL CREATE TABLE statement to create the spatial index table.
- For performance reasons, create an index on the SDO_GID column in the <layername>_SDOGEOM table before calling this routine.
- This procedure generates either fixed-size or variable-sized tiles depending on values stored in the <layername>_SDOLAYER table as follows:

SDO_LEVEL	SDO_NUMTILES	Action
NULL	NULL	Error
>= 1	NULL	Fixed-size tiling

SDO_LEVEL	SDO_NUMTILES	Action
>= 1	>= 1	Indexing with variable-sized tiles. The SDO_LEVEL column defines the partition bucket size. The SDO_NUMTILES column defines the number of tiles to generate per geometry. Note: variable-sized tiling is for experimentation purposes only.
NULL	>= 1	Not supported

- If the <layername>_SDOINDEX table is empty, the procedure selects all the geometries in the geometry table and generates index entries for them. If the index table is not empty, the procedure determines which entries in the geometry table do not have index entries, and generates them.
- `SDO_ADMIN.POPULATE_INDEX()` behaves similarly to the CREATE INDEX statement in SQL. An implicit commit is executed after the procedure is called.
- `SDO_ADMIN.POPULATE_INDEX()` operates as a single transaction. To reduce the amount of rollback required to execute this procedure, you can write a routine that loops and calls `SDO_ADMIN.UPDATE_INDEX()` repeatedly. See Section A.1.1.1, “cr_spatial_index.sql Script” for more information.

Example 5–1 tessellates all the geometric objects in the LAYER1_SDOGEOM table and adds the generated tiles to the LAYER1_SDOINDEX table.

Example 5–1

```
SQL> EXECUTE SDO_ADMIN.POPULATE_INDEX('layer1');
SQL> COMMIT;
```

Related Topics

- SDO_ADMIN.UPDATE_INDEX() procedure

SDO_ADMIN.POPULATE_INDEX_FIXED

Purpose

This procedure is provided for compatibility with Spatial Cartridge release 8.0.3 tables. This procedure has been replaced by enhanced features in the `SDO_ADMIN.POPULATE_INDEX()` procedure, and by supporting schema changes as shown in Section 1.4.

This procedure tessellates a list of geometric objects created by selecting all the entries in the geometry table that do not have corresponding entries in the spatial index table. This procedure can also tessellate all the geometric objects in a geometry table or view and add the tiles to the spatial index table.

Use this procedure to tessellate the geometries into fixed-size tiles.

Syntax

SDO_ADMIN.POPULATE_INDEX_FIXED (*layername, tile_size, [synch_flag,] [sdo_tile_flag,] [sdo_maxcode_flag])*

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
tile_size	Specifies the number of tessellations required to achieve the desired tile size (see the Usage Notes). Data type is INTEGER.
synch_flag	Specifies whether to tessellate every geometric object in the geometry table, or only those that do not have corresponding entries in the spatial index table. If TRUE, only those geometric objects in the geometry table that do not have any corresponding tiles in the spatial index table are tessellated. If FALSE, all the geometric objects in the geometry table are tessellated and new tiles are simply added to the spatial index table. Default value is TRUE. Data type is BOOLEAN.
sdo_tile_flag	For internal use only. Not supported in this release. Default value is FALSE.

sdo_maxcode_flag	Specifies whether or not the SDO_MAXCODE column is populated. If TRUE, SDO_MAXCODE is populated. If FALSE, the column is not populated. Default value is TRUE. Data type is BOOLEAN.

Usage Notes

Note: This procedure is likely to be removed in a future release of Spatial Cartridge.

Consider the following when using this procedure:

- The SQL CREATE TABLE statement is used to create the spatial index table, <layername>_SDOINDEX, prior to calling this procedure.
- The layer is tessellated into equal-sized tiles based on the number passed in the tile_size parameter. The value of tile_size specifies how many times to tessellate the layer. See Section 2.3.2, “Spatial Indexing with Fixed-Size Tiles”.
- For performance reasons, set the synch_flag to FALSE when the spatial index table contains zero rows.
- For performance reasons, create an index on the SDO_GID column in the <layername>_SDOGEOM table before calling this routine.
- `SDO_ADMIN.POPULATE_INDEX_FIXED()` behaves similarly to the CREATE INDEX statement in SQL. An implicit commit is executed after the procedure is called.
- `SDO_ADMIN.POPULATE_INDEX_FIXED()` operates as a single transaction. To reduce the amount of rollback required to execute this procedure, you can write a routine that loops and calls `SDO_ADMIN.UPDATE_INDEX_FIXED()` repeatedly. See Section A.1.1.1, “cr_spatial_index.sql Script” for more information.

Example 5–2 tessellates all the geometric objects in the LAYER1_SDOGEOM table using up to 256 (4^4) fixed-size tiles and adds the generated tiles to the LAYER1_SDOINDEX table.

Example 5–2

```
SQL> EXECUTE SDO_ADMIN.POPULATE_INDEX_FIXED('layer1',4,FALSE,FALSE,FALSE);
```

Related Topics

- SDO_ADMIN.UPDATE_INDEX_FIXED() procedure
- SDO_TUNE.ESTIMATE_TILING_LEVEL() function

SDO_ADMIN.SDO_CODE_SIZE

Purpose

This function determines the size that the SDO_CODE column should be in the <layername>_SDOINDEX table.

Syntax

SDO_ADMIN.SDO_CODE_SIZE (*layername*)

Keywords and Parameters

layername	Specifies the name of the data set layer. Data type is VARCHAR2.

Returns

This function returns the required size in bytes for the SDO_CODE column. Data type is INTEGER.

Usage Notes

The SDO_CODE column is used to store the bit-interleaved cell ID of a tile that covers a geometry. The SDO_MAXCODE column is SDO_CODE padded out one place farther than the longest allowable code name for the index. Both columns are defined as RAW data types, with a maximum of 255 bytes. Use the `SDO_ADMIN.SDO_CODE_SIZE()` function to fine-tune the size of the columns.

You should always set the SDO_MAXCODE column to one greater than the SDO_CODE column.

Related Topics

None

SDO_ADMIN.UPDATE_INDEX

Purpose

This procedure tessellates a single geometric object in a geometry table or view and adds the tiles to the spatial index table. If the object already exists and has index entries, those entries are deleted and replaced by the newly generated tiles.

Syntax

SDO_ADMIN.UPDATE_INDEX (*layername, GID*)

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry table. Data type is VARCHAR2.
GID	Specifies the geometric object identifier. Data type is NUMBER.

Usage Notes

Considert the following when using this procedure:

- The <layername>_SDOINDEX table must exist prior to calling this procedure. Use the SQL CREATE TABLE statement to create the spatial index table.
- For performance reasons, create an index on the SDO_GID column in the <layername>_SDOGEOM table before calling this routine.
- The values of the SDO_LEVEL and SDO_NUMTILES columns must be set in the <layername>_SDOLAYER table before calling this procedure. This procedure generates either fixed-size or variable-sized tiles depending on values stored in the <layername>_SDOLAYER table as follows:

SDO_LEVEL	SDO_NUMTILES	Action
NULL	NULL	Error
>= 1	NULL	Fixed-size tiling

SDO_LEVEL	SDO_NUMTILES	Action
>= 1	>= 1	Indexing with variable-sized tiles. The SDO_LEVEL column defines the partition bucket size. The SDO_NUMTILES column defines the number of tiles to generate per geometry. Note: variable-sized tiling is for experimentation purposes only.
NULL	>= 1	Not supported

- `SDO_ADMIN.UPDATE_INDEX()` does not perform an implicit commit after it executes and therefore the transaction can be rolled back.

Example 5–3 tessellates the polygon for geometry 25 and adds the generated tiles to the LAYER1_SDOINDEX table.

Example 5–3

```
SQL> EXECUTE SDO_ADMIN.UPDATE_INDEX('layer1', 25);
SQL> COMMIT;
```

Related Topics

- SDO_ADMIN.POPULATE_INDEX() procedure

SDO_ADMIN.UPDATE_INDEX_FIXED

Purpose

This procedure is provided for compatibility with Spatial Cartridge release 8.0.3 tables. This procedure has been replaced by enhanced features in the `SDO_ADMIN.UPDATE_INDEX()` procedure, and by supporting schema changes as shown in Section 1.4.

This procedure tessellates a single geometric object in a geometry table or view and adds the fixed-sized tiles to the spatial index table. By default, these tiles will replace existing ones for the same geometry; or optionally, existing tiles can be left alone.

Syntax

SDO_ADMIN.UPDATE_INDEX_FIXED (*layername, GID, tile_size, [replace_flag,] [sdo_tile_flag] [sdo_maxcode_flag]*)

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry table. Data type is VARCHAR2.
GID	Specifies the geometric object identifier. Data type is NUMBER.
tile_size	Specifies the number of tessellations required to achieve the desired fixed-size tiles. Each tessellation subdivides the tiles from the previous level into four smaller tiles. Data type is INTEGER.
replace_flag	Specifies whether or not to delete tiles for the GID before adding new ones. If TRUE, tiles are deleted prior to inserting new entries into the spatial index table. If FALSE, new tiles are simply added to the spatial index table. Default value is TRUE. Data type is BOOLEAN.
sdo_tile_flag	For internal use only. Not supported in this release. Default value is FALSE. Data type is BOOLEAN.

sdo_maxcode_flag	Specifies whether or not the SDO_MAXCODE column is populated. If TRUE, SDO_MAXCODE is populated. If FALSE, the column is not populated. Default value is TRUE. Data type is BOOLEAN.

Usage Notes

Note: This procedure is likely to be removed in a future release of Spatial Cartridge.

Consider the following when using this procedure:

- For performance reasons, set the replace_flag to FALSE when the spatial index table contains no entries for the specified GID.
- For performance reasons, create an index on the SDO_GID column in the <layername>_SDOGEOM table before calling this procedure.
- `SDO_ADMIN.UPDATE_INDEX_FIXED()` does not perform an implicit commit after it executes and therefore this transaction can be rolled back.

Example 5–4 tessellates the polygon for geometry 25 and adds the generated tiles to the LAYER1_SDOINDEX table.

Example 5–4

```
SQL> EXECUTE SDO_ADMIN.UPDATE_INDEX_FIXED ('layer1',25,4,FALSE,FALSE,FALSE);
```

Related Topics

- SDO_ADMIN.POPULATE_INDEX_FIXED() procedure
- SDO_TUNE.ESTIMATE_TILING_LEVEL() function

SDO_ADMIN.VERIFY_LAYER

Purpose

This procedure checks for the existence of the geometry and spatial index tables.

Syntax

SDO_ADMIN.VERIFY_LAYER (*layername,[maxtiles]*)

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
maxtiles	For internal use only. Not supported in this release.

Usage Notes

If this procedure does not find the geometry and spatial index tables, it generates the following error: SDO 13113 (Oracle table does not exist)

Example 5–5 verifies the LAYER1 data set layer:

Example 5–5

```
SQL> EXECUTE SDO_ADMIN.VERIFY_LAYER('layer1');
```

Related Topics

None

Partitioned Point Data Procedures

Table 5–2 lists the procedures that can be used with partitioned point data. These procedures are neither required nor compatible with the geometry-based data format.

Table 5–2 Administrative Procedures for Partitioned Point Data

Procedure	Data Structure	Description
SDO_ADMIN.ALTER_HIGH_WATER_MARK	Partitioned points	Alters the high-water mark of a partitioned table.
SDO_ADMIN.DROP_PARTITION_INFO	Partitioned points	Removes a partitioned table.
SDO_ADMIN.PARTITION	Partitioned points	Places data into partition tables.
SDO_ADMIN.PROPAGATE_GRANTS	Partitioned points	Propagates the grants on the registered underlying table to the various partitions.
SDO_ADMIN.REGISTER_PARTITION_INFO	Partitioned points	Creates a partitioned spatial table.
SDO_ADMIN.REPARTITION	Partitioned points	Reorganizes a table based on the sorted values of the data contained within it.
SDO_ADMIN.VERIFY_PARTITIONS	Partitioned points	Checks for the existence of a table.

Also see Appendix A, "Sample SQL Scripts and Tuning Tips" for additional administrative tools useful for working with partitioned point data.

SDO_ADMIN.ALTER_HIGH_WATER_MARK

Purpose

This procedure alters the high-water mark of a partitioned spatial table. The high-water mark defines how many records can be stored in a partition before it subdivides. The table must exist and be registered in the Spatial Cartridge data dictionary.

This procedure is for use only with partitioned point data.

Syntax

SDO_ADMIN.ALTER_HIGH_WATER_MARK (*tablename*, *high_water_mark*)

Keywords and Parameters

tablename	Specifies the name of the partitioned table. Data type is VARCHAR2.
high_water_mark	Specifies the new high-water mark for the table. Data type is INTEGER.

Usage Notes

None

Example 5–6 changes the high-water mark to 5000 records for the TABLE1 partitioned spatial table.

Example 5–6

```
SQL> EXECUTE SDO_ADMIN.ALTER_HIGH_WATER_MARK('table1', 5000);
```

Related Topics

- SDO_ADMIN.REPARTITION() procedure
- altpart.sql sample SQL script file

SDO_ADMIN.DROP_PARTITION_INFO

Purpose

This procedure removes a partitioned spatial table from the Spatial Cartridge data dictionary. The table must exist and must be registered in the Spatial Cartridge data dictionary.

This procedure is used only with partitioned point data.

Syntax

SDO_ADMIN.DROP_PARTITION_INFO (*tablename*)

Keywords and Parameters

tablename	Specifies the name of the partitioned table. Data type is VARCHAR2.

Usage Notes

This procedure does not remove the spatial table and its associated partition tables from the user's schema. For a description of how to remove a partitioned spatial table from the user's schema, see the drppart.sql sample SQL script file described in Section A.1.2.2.

Example 5–7 removes the table1 table from the Spatial Cartridge data dictionary.

Example 5–7

```
SQL> EXECUTE SDO_ADMIN.DROP_PARTITION_INFO('table1');
```

Related Topics

- drppart.sql sample SQL script file

SDO_ADMIN.PARTITION

Purpose

This procedure places data into partition tables based on the sorted order of encoded dimensional values.

This procedure is used only with partitioned point data.

Syntax

SDO_ADMIN.PARTITION (*owner.source_table, tablename, parallel, guess , plummet_flag [,tablespace]*)

Keywords and Parameters

owner.source_table	Specifies the Oracle8 table or view of the table containing the partition key column. Data type is VARCHAR2.
tablename	Specifies the name of the table to partition. Data type is VARCHAR2.
parallel	Specifies the degree of parallelism for an operation on a single instance. Data type is INTEGER.
guess	Specifies the estimated largest common level of all the potential partitions to be created from data in the source_table. The common level of a partition is the number of levels of resolution of the common HHCODE for the partition. Data type is INTEGER.
plummet_flag	Specifies if the common HHCODE for all the potential partitions to be created from data in the source_table contains the maximum possible common level. If TRUE, the common HHCODE for each potential partition contains the maximum possible common level. If FALSE, the common HHCODE for each potential partition contains the minimum possible common level. Default value is FALSE. Data type is BOOLEAN.
tablespace	Specifies the tablespace in which the partitions should be created. Default is the tablespace of the underlying table.

Usage Notes

Consider the following when using this procedure:

- The maximum size of the partition tables is determined by the high-water mark of the partitioned spatial table.
- To perform this procedure, first load the original data into an Oracle8 table using a utility such as SQL*Loader. After the data is loaded, encode the data using the appropriate combination of Spatial Cartridge data conversion functions (see Chapter 9.) The encoded data is used as the partition key column. The partition key column is provided as either a column in the Oracle8 table or as a view of that table.
- For more information on specifying the degree of parallelism, see the *Oracle8 Server Tuning* manual.

Example 5–8 partitions the table1 partitioned spatial table with data contained in the source1 table.

Example 5–8

```
SQL> EXECUTE SDO_ADMIN.PARTITION('source1','table1',1,10,FALSE);
```

Related Topics

- SDO_ADMIN.REGISTER_PARTITION_INFO() procedure

SDO_ADMIN.PROPAGATE_GRANTS

Purpose

This procedure is used to propagate the grants on the underlying table to the partitions.

This procedure is used only with partitioned point data.

Syntax

SDO_ADMIN.PROPAGATE_GRANTS (*tablename)*

Keywords and Parameters

tablename	Specifies the name of the partitioned table. Data type is VARCHAR2.

Usage Notes

This procedure is used after calls to `SDO_ADMIN.PARTITION()` or `SDO_ADMIN.REPARTITION()`. It must be called by the owner of the partition.

This procedure must be compiled prior to use. See Section A.1.2.3, “sdogrant.sql Script”.

Example 5–9 propagates grants from the TABLE1 partitioned spatial table.

Example 5–9

```
SQL>  EXECUTE SDO_ADMIN.PROPAGATE_GRANTS('TABLE1');
```

Related Topics

- SDO_ADMIN.PARTITION() procedure
- SDO_ADMIN.REPARTITION() procedure

SDO_ADMIN.REGISTER_PARTITION_INFO

Purpose

This procedure creates a partitioned spatial table entry in the Spatial Cartridge data dictionary, and defines the partition key column and the high-water mark for the table.

This procedure is used only with partitioned point data.

Syntax

SDO_ADMIN.REGISTER_PARTITION_INFO (*tablename, column, high_water_mark*)

Keywords and Parameters

tablename	Specifies the name of the partitioned table. Data type is VARCHAR2.
column	Specifies the name of the partition key column for the table. Data type is VARCHAR2.
high_water_mark	Specifies the number of records to store in a partition before the partition subdivides. Data type is INTEGER.

Usage Notes

The SQL CREATE TABLE statement is used to create the partitioned spatial table, with the partition key column defined as RAW(255), prior to calling this procedure.

Example 5–10 registers the TABLE1 partitioned spatial table.

Example 5–10

```
SQL> EXECUTE SDO_ADMIN.REGISTER_PARTITION_INFO('table1',
2> 'hhcolumn', 1000);
```

Related Topics

- SDO_ADMIN.PARTITION() procedure

SDO_ADMIN.REPARTITION

Purpose

This procedure reorganizes a partitioned spatial table based on the sorted order of encoded dimensional values already contained in it. The table must exist and must be registered in the Spatial Cartridge data dictionary.

This procedure is used only with partitioned point data.

Syntax

SDO_ADMIN.REPARTITION (*tablename, parallel, [tablespace]*)

Keywords and Parameters

tablename	Specifies the name of the partitioned table. Data type is VARCHAR2.
parallel	Specifies the degree of parallelism for an operation on a single instance. Data type is INTEGER.
tablespace	Specifies the name of the tablespace in which to create the partition. Data type is VARCHAR2.

Usage Notes

Consider the following when using this procedure:

- The tablespace variable is optional. If you do not supply a tablespace name, the partitions are created in the same tablespace as the registered partition table.
- The maximum size of the reorganized partition tables is determined by the high-water mark of the partitioned spatial table.
- For more information on specifying the degree of parallelism, see the section on "Parallel Query Option," in the Oracle8 Server documentation.

Example 5–11 repartitions the table1 partitioned spatial table.

Example 5–11

```
SQL> EXECUTE SDO_ADMIN.REPARTITION('table1', 1);
```

Related Topics

- SDO_ADMIN.ALTER_HIGH_WATER_MARK() procedure

SDO_ADMIN.VERIFY_PARTITIONS

Purpose

This procedure checks if the partitioned spatial table exists, if it is registered in the Spatial Cartridge data dictionary, and if the partition key column exists as defined in the Spatial Cartridge data dictionary.

This procedure is used only with partitioned point data.

Syntax

SDO_ADMIN.VERIFY_PARTITIONS (*tablename*)

Keywords and Parameters

tablename	Specifies the name of the table. Data type is VARCHAR2.

Usage Notes

This procedure can generate the following errors depending on the results of the verification:

- SDO 13113 (Oracle table does not exist)
- SDO 13108 (spatial table not found)
- SDO 13111 (spatial table has no partition key defined)
- SDO 13129 (HHCODE column not found)

Example 5–12 verifies the TABLE1 partitioned spatial table:

Example 5–12

```
SQL> EXECUTE SDO_ADMIN.VERIFY_PARTITIONS('table1');
```

Related Topics

- SDO_ADMIN.REGISTER_PARTITION_INFO() procedure

6 Tuning Functions

This chapter contains descriptions of the tuning functions and procedures shown in Table 6–1.

Table 6–1 Tuning Functions and Procedures:

Function/Procedure	Description
SDO_TUNE.ESTIMATE_TILING_LEVEL	Determines an appropriate tiling level for creating fixed-size index tiles.
SDO_TUNE.EXTENT_OF	Determines the minimum-bounding rectangle of the data in a layer.

SDO_TUNE.ESTIMATE_TILING_LEVEL

Purpose

This function estimates the appropriate tiling level to use when indexing with fixed-size tiles.

Syntax

SDO_TUNE.ESTIMATE_TILING_LEVEL *(layername, maxtiles, type_of_estimate)*

Keywords and Parameters

layername	Specifies the name of the data set layer to examine. Data type is VARCHAR2.
maxtiles	Specifies the maximum number of tiles that can be used to index the rectangle defined by the type_of_estimate parameter. Data type is INTEGER.
type_of_estimate	Indicates by keyword one of three different models. Specify the type of estimate with one of the following keywords: • LAYER_EXTENT -- Use the rectangle defined by your coordinate system. • ALL_GID_EXTENT -- Use the minimum-bounding rectangle that encompasses all the geometric objects within the layer. • AVG_GID_EXTENT -- Use a rectangle representing the average size of the individual geometries within the layer. This option performs the most analysis of the three types.

Returns

The function returns an integer representing the level to use when creating a spatial index for the specified layer.

Usage Notes

The `SDO_ADMIN.POPULATE_INDEX()` and `SDO_ADMIN.UPDATE_INDEX()` procedures are used to create or update the spatial index using fixed-size or variable-sized tiles. Store the value returned by the `SDO_TUNE.ESTIMATE_TILING_LEVEL()` function in the SDO_LEVEL column of the <layername>_SDOLAYER table prior to building the spatial index.

The maxtiles parameter specifies the maximum number of tiles that should be used to define a grid covering the rectangular extend of interest. This extent could be:

- Defined in the <layername>_SDODIM table which defines the bounds of the coordinate system
- Defined by the minimum and maximum coordinates for the given data set (as returned by the `SDO_TUNE.EXTENT_OF()` procedure)
- Defined by computing the average bounds of the objects in the <layername>_SDOGEOM table

The code shown in Example 6–1 generates a recommendation based on the extent of the defined coordinate system (-90 to +90 latitude and -180 to +180 longitude). This example returns a level whose tiles are not smaller than one-degree cells.

Example 6–1 Recommended Tile Level for One-Degree Lat/Lon Cells

```
set serveroutput on
declare
   tiling_level integer;
begin
   tiling_level := mdsys.sdo_tune.estimate_tiling_level('WORLD_CITIES',
           360*180, 'LAYER_EXTENT');
    dbms_output.put_line('VALUE is '|| tiling_level);
end;
```

For many applications, however, it is more effective to call the `SDO_TUNE.ESTIMATE_TILING_LEVEL()` function using the ALL_GID_EXTENT estimate type. In Example 6–2, assume the data set consists of block groups for San Francisco and that the <layername>_SDODIM table defines the extent to be one that covers all of California. Because the data set is localized to a small subregion of this extent, ALL_GID_EXTENT is the appropriate estimate type. The recommended tiling level in this case will be such that at most 10,000 tiles will be required to completely cover the extent of San Francisco block groups.

Example 6–2 Recommended Tile Level Based on the GIDs of All Geometries

```
set serveroutput on
declare
   tiling_level integer;
```

```
begin
   tiling_level:=   mdsys.sdo_tune.estimate_tiling_level('SF_BLOCK_GROUPS',
         10000, 'ALL_GID_EXTENT');
   dbms_output.put_line('VALUE is' ,|| tiling_level);
end;
```

The third type of estimate helps determine the tiling level that should be used such that on average, the maxtiles parameter defines the number of tiles to cover the extent of a single geometry in the layer. This estimate type requires the most computation of the three because the bounding rectangle of every geometry is used in calculating the average extent. In Example 6–3, eight tiles on average are used to cover any block group in San Francisco.

Example 6–3 Recommended Tile Level Based on Average Extent of All Geometries

```
set serveroutput on
declare
   tiling_level integer;
begin
   tiling_level := mdsys.sdo_tune.estimate_tiling_level('SF_BLOCK_GROUPS', 8,
         'AVG_GID_EXTENT');
   dbms_output.put_line('Tiling level value is ' || tiling_level);
end;
```

Related Topics

- SDO_ADMIN.POPULATE_INDEX
- SDO_ADMIN.UPDATE_INDEX
- SDO_TUNE.EXTENT_OF
- Section A.2.2, “Understanding the Tiling Level”
- Section A.2.8, “Visualizing the Spatial Index (Drawing Tiles)”

SDO_TUNE.EXTENT_OF

Purpose

This procedure determines the extent of all geometries in a layer.

Syntax

SDO_TUNE.EXTENT_OF *(layername, min_X, max_X, min_Y, max_Y)*

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
min_X	Minimum X value of the bounding rectangle. Data type is NUMBER.
max_X	Maximum X value of the bounding rectangle. Data type is NUMBER.
min_Y	Minimum Y value of the bounding rectangle. Data type is NUMBER.
max_Y	Maximum Y value of the bounding rectangle. Data type is NUMBER.

Returns

This procedure returns the coordinates of the minimum-bounding rectangle for all geometric data in a layer. The data type is NUMBER for the four return values.

Usage Notes

None

Related Topics

- SDO_TUNE.ESTIMATE_TILING_LEVEL() function

7 Geometry Functions

This chapter contains descriptions of the geometric functions and procedures shown in Table 7–1.

Table 7–1 Geometric Functions and Procedures

Function/Procedure	Description
SDO_GEOM.ADD_NODES	Stores points in the geometry table.
SDO_GEOM.INIT_ELEMENT	Initializes space in the geometry table for a new object.
SDO_GEOM.INTERACT	Determines if two objects are disjoint.
SDO_GEOM.RELATE	Determines how two objects interact.
SDO_GEOM.VALIDATE_GEOMETRY	Determines if a geometry is valid.

SDO_GEOM.ADD_NODES

Purpose

This procedure stores coordinate geometry points into the SDOGEOM table.

Syntax

SDO_GEOM.ADD_NODES *(layername, SDO_GID, SDO_ESEQ, SDO_ETYPE, X-ord1,Y-ord1[,...,X_{125}, Y_{125}])*

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
SDO_GID	Specifies the unique geometric object identifier. Data type is NUMBER.
SDO_ESEQ	Specifies the element sequence number. The eseq is unique within an SDO_GID. Data type is NUMBER.
SDO_ETYPE	Specifies the type of geometric element. Data type is INTEGER, corresponding to the following constants: 1 SDO_GEOM.POINT_TYPE 2 SDO_GEOM.LINESTRING_TYPE 3 SDO_GEOM.POLYGON_TYPE
X ordinateN, *Y ordinateN*	Specifies the X and Y values of a vertex (coordinate pair) in a geometry. Up to 125 pairs may be added in a single call. Data type is NUMBER.

Usage Notes

Consider the following when using this procedure:

- Use the SQL CREATE TABLE statement to create the geometry table, <layername>_SDOGEOM, before calling this procedure.

- Prior to calling this procedure, call the SDO_GEOM.INIT_ELEMENT() function to initialize the geometry element and retrieve the element sequence number (SDO_ESEQ).
- Close a polygon by providing the coordinates of the first vertex as the last vertex.
- Call this procedure iteratively with the same GID to add coordinates to a geometric object. You can add up to 125 coordinate pairs on each call, and there is no limit to how many times you can add more vertices.
- This procedure cannot be used when <layername>_SDOGEOM is a view. For layer objects created as views, you need to explicitly insert new geometries, as opposed to using `SDO_GEOM.INIT_ELEMENT()` and `SDO_GEOM.ADD_NODES()`.

Example 7–1 adds a polygon element for geometry 25 in the LAYER1 data set. The polygon is a square.

Example 7–1

```
SQL> EXECUTE SDO_GEOM.ADD_NODES ('LAYER1', 25,
2> SDO_GEOM.POLYGON_TYPE,
3> 3,3, 7,3,
4> 7,7, 3,7,
5> 3,3);
```

Related Topics

- SDO_GEOM.INIT_ELEMENT() function

SDO_GEOM.INIT_ELEMENT

Purpose

This function initializes elements in the SDOINFO table or view for a new geometry element.

Syntax

SDO_GEOM.INIT_ELEMENT (*layername, SDO_GID*)

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
SDO_GID	Specifies the geometric object identifier. Data type is NUMBER.

Returns

This function returns the next element sequence number. The data type is INTEGER.

Usage Notes

This function initializes the element to be stored, but does not actually insert coordinates into the SDOGEOM table. The `SDO_GEOM.ADD_NODES()` procedure is used to insert associated coordinate data.

For layer objects created as views, you need to explicitly insert new geometries, as opposed to using `SDO_GEOM.INIT_ELEMENT()` and `SDO_GEOM.ADD_NODES()`.

Related Topics

- SDO_GEOM.ADD_NODES() procedure

SDO_GEOM.INTERACT

Purpose

This function determines if two geometry objects interact.

Syntax

SDO_GEOM.INTERACT (*layername1, SDO_GID1, [layername2,] SDO_GID2*)

SDO_GEOM.INTERACT (*layername1, SDO_GID1, X_tolerance, Y_tolerance, SDO_ETYPE, num_ordinates, X_ordinate1, Y_ordinate1 [, ..., Xn, Yn] [,SDO_ETYPE, num_ordinates, X_ordinate1, Y_ordinate1 [,...Xn,Yn]]*)

Keywords and Parameters

layername1, layername2	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
SDO_GID1, SDO_GID2	Specifies the geometric object identifier. Data type is NUMBER.
X_tolerance, Y_tolerance	Specifies the distance two points can be apart and still be considered the same due to rounding errors. Tolerance must be greater than zero. If you want zero tolerance, enter a number such as 0.000005, where the number of zeroes to the right of the decimal point matches the precision of your data. Data type is NUMBER.
SDO_ETYPE	Specifies the type of geometry object. Data type is INTEGER, corresponding to the following constants:

1	SDO_GEOM.POINT_TYPE
2	SDO_GEOM.LINESTRING_TYPE
3	SDO_GEOM.POLYGON_TYPE

num_ordinates	Specifies the number of ordinates for this element. Data type is NUMBER.
X_ordinateN, Y_ordinateN	Specifies the X and Y values of a vertex (coordinate pair) in a geometry. Data type is NUMBER.

Returns

This function returns TRUE if the first and second objects interact with each other and are not disjoint. The data type is VARCHAR2.

Usage Notes

Note: The `SDO_GEOM.INTERACT()` procedure has been replaced by the `SDO_GEOM.RELATE()` procedure with the ANYINTERACT keyword. This procedure may be removed in a future version of Spatial Cartridge.

Use the first form of the function to test two stored geometry objects.

Use the second form of the function to compare a stored object against a user-defined object. You can specify up to 123 vertices for a single element geometry. If the geometry has multiple elements, the total number of arguments passed, including SDO_ETYPE, num_ordinates, and the list of vertex coordinates cannot exceed 250 values.

Related Topics

- SDO_GEOM.RELATE() function

SDO_GEOM.RELATE

Purpose

This function examines two geometry objects to determine their spatial relationship.

Syntax

SDO_GEOM.RELATE (*layername1, SDO_GID1, mask, [layername2,] SDO_GID2*)

SDO_GEOM.RELATE (*layername1, SDO_GID1, mask, X_tolerance, Y_tolerance, SDO_ETYPE, num_ordinates, X_ordinate1, Y_ordinate1 [,...,Xn, Yn] [,SDO_ETYPE, num_ordinates, X_ordinate1, Y_ordinate1 [,...,Xn, Yn]])*

Keywords and Parameters

layername1, layername2	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
SDO_GID1, SDO_GID2	Specifies the geometry object identifier. Data type is NUMBER.
mask	Specifies a list of relationships to check. See the list of keywords in the Usage Notes.
X_tolerance, *Y_tolerance*	Specifies the distance two points can be apart and still be considered the same due to rounding errors. Tolerance must be greater than zero. If you want zero tolerance, enter a number such as 0.000005, where the number of zeroes to the right of the decimal point matches the precision of your data. Data type is NUMBER.
SDO_ETYPE	Specifies the type of geometry element. Data type is INTEGER, corresponding to the following constants:

1	SDO_GEOM.POINT_TYPE
2	SDO_GEOM.LINESTRING_TYPE
3	SDO_GEOM.POLYGON_TYPE

num_ordinates	Specifies the number of ordinates for this element. Data type is NUMBER.
X_ordinateN, Y_ordinateN	Specifies the X and Y values of a vertex (coordinate pair) in a geometry. Data type is NUMBER.

Returns

The `SDO_GEOM.RELATE()` function can return three types of answers:

1. If you pass a mask listing one or more relationships, the function returns the names of the relationships if all of them are true. If one or more relationships are false, the procedure returns FALSE.
2. If you pass the DETERMINE keyword in the mask, the function returns the one relationship keyword that best matches the geometries.
3. If you pass the ANYINTERACT keyword in the mask, the function returns TRUE if the two geometries are not disjoint. This is equivalent to the `SDO_GEOM.INTERACT()` function.

The data type is VARCHAR2.

Usage Notes

Use the first form of the function to examine two stored geometry objects.

Use the second form of the function to compare a stored object against a user-defined object. You can specify up to 123 vertices for a single-element geometry. If the geometry has multiple elements, the total number of arguments passed, including SDO_ETYPE, num_ordinates, and the list of vertex coordinates cannot exceed 250 values.

The following relationships can be tested:

- ANYINTERACT - Returns TRUE if the objects are not disjoint.
- CONTAINS - Returns TRUE if the second object is entirely within the first object and the object boundaries do not touch.
- COVEREDBY - Returns TRUE if the first object is entirely within the second object and the object boundaries touch at one or more points.
- COVERS - Returns TRUE if the second object is entirely within the first object and the boundaries touch in one or more places.

- DISJOINT - Returns TRUE if the objects have no common boundary or interior points.
- EQUAL - Returns TRUE if the objects share every point of their boundaries and interior, including any holes in the objects.
- INSIDE - Returns TRUE if the first object is entirely within the second object and the object boundaries do not touch.
- OVERLAPBDYDISJOINT - Returns TRUE if the objects overlap, but their boundaries do not interact.
- OVERLAPBDYINTERSECT - Returns TRUE if the object overlap, and their boundaries intersect in one or more places.
- TOUCH - Returns TRUE if the two objects share a common boundary point, but no interior points.

Mask values can be combined using a logical OR. For example, 'INSIDE + TOUCH' returns TRUE if the objects pass either test.

Related Topics

- SDO_GEOM.INTERACT() function

SDO_GEOM.VALIDATE_GEOMETRY

Purpose

This function provides a consistency check for valid geometry types. The function checks the representation of the geometry from the tables against the element definitions.

Syntax

SDO_GEOM.VALIDATE_GEOMETRY (*layername,SDO_GID*)

Keywords and Parameters

layername	Specifies the name of the data set layer. The layer name is used to construct the name of the geometry and spatial index tables. Data type is VARCHAR2.
SDO_GID	Specifies the geometric object identifier. Data type is NUMBER.

Returns

This function returns TRUE if the geometry is valid. The data type is VARCHAR2.

Usage Notes

This function checks for the following:

- Polygons have at least three points and must be closed
- Line strings must have at least two points
- When an SDO_ESEQ spans multiple rows, the last point of the previous row is the first point on the next row

Related Topics

None

8 Window Functions

If a query window does not already exist in the database, you must first insert it and create and index for it. The SDO_WINDOW functions are used to create temporary geometry objects to be used in comparisons with stored geometries. You can create query windows with any number of coordinates.

Because not all Oracle users may have insert privileges, the SDO_WINDOW package is not automatically installed when you install Spatial Cartridge. This allows a DBA to control the schema under which these functions operate. Choose an Oracle user who has insert privilege and compile the SDO_WINDOW package under that user. For example, you could choose the mdsys Oracle user:

```
% sqlplus mdsys/password
SQL> @$ORACLE_HOME/md/admin/sdowin.sql
SQL> @$ORACLE_HOME/md/admin/prvtwin.plb
```

This chapter contains descriptions of the window functions listed in Table 8–1.

Table 8–1 Window Functions

Function	Description
SDO_WINDOW.BUILD_WINDOW	Builds a query window geometry object.
SDO_WINDOW.BUILD_WINDOW_FIXED	Builds a query window using fixed-size tiles.
SDO_WINDOW.CLEAN_WINDOW	Drops the tables used for a query window.
SDO_WINDOW.CLEANUP_GID	Removes the query window without dropping the tables.
SDO_WINDOW.CREATE_WINDOW_LAYER	Creates the tables needed for a query window layer.

SDO_WINDOW.BUILD_WINDOW

Purpose

This function builds the window for the query and returns an SDO_GID that serves as a handle. The window is tessellated into variable-sized tiles.

Syntax

SDO_WINDOW.BUILD_WINDOW(*comp_name*, *layername*, *SDO_ETYPE*, *SDO_NUMTILES*, *X1*, *Y1*, *[...Xn*, *Yn]*)

Keywords and Parameters

comp_name	Specifies the name of the user who compiled this package. This user must have appropriate privileges to read and write into the database. Data type is VARCHAR2.
layername	Specifies the name of the window layer into which the coordinates will be inserted. Data type is VARCHAR2.
SDO_ETYPE	Specifies the type of geometry element. Data type is INTEGER, corresponding to the following constants: 1 SDO_GEOM.POINT_TYPE 2 SDO_GEOM.LINESTRING_TYPE 3 SDO_GEOM.POLYGON_TYPE
SDO_NUMTILES	Value must be NULL for Spatial Cartridge release 8.0.4 and later. Data type is NUMBER.
X ordinateN, *Y ordinateN*	Specifies the X and Y values of a vertex (coordinate pair) in a geometry. Up to 125 pairs may be added in a single call. Data type is NUMBER.

Returns

This function returns the SDO_GID of the new geometry. The data type is NUMBER.

Usage Notes

This function inserts the coordinates into the <layername>_SDOGEOM table, tessellates the geometry (creates the index), and returns a unique SDO_GID corresponding to the geometry.

You do not need special privileges to execute this function. However, the user who compiles it does need appropriate privileges to read and write into the database.

When working with Spatial Cartridge release 8.0.3 tables, the SDO_NUMTILES parameter indicates the number of tiles into which the window should be tessellated. For release 8.0.4 and later, the function reads that information from the <layername>_SDOLAYER table.

Related Topics

- SDO_WINDOW.BUILD_WINDOW_FIXED() function

SDO_WINDOW.BUILD_WINDOW_FIXED

Purpose

This function builds the window for the query and returns an SDO_GID that serves as a handle.   The window is tessellated into fixed-size tiles.

Syntax

SDO_WINDOW.BUILD_WINDOW_FIXED (*comp_name, layername, SDO_ETYPE, SDO_TILESIZE, X1, Y1, [...Xn, Yn]*)

Keywords and Parameters

comp_name	Specifies the name of the user who compiled this package. This user must have appropriate privileges to read and write into the database. Data type is VARCHAR2.
layername	Specifies the name of the window layer into which the coordinates will be inserted. Data type is VARCHAR2.
SDO_ETYPE	Specifies the type of geometry element. Data type is INTEGER, corresponding to the following constants:

1	SDO_GEOM.POINT_TYPE
2	SDO_GEOM.LINESTRING_TYPE
3	SDO_GEOM.POLYGON_TYPE

SDO_TILESIZE	Specifies the number of tessellations required to achieve the desired fixed-size tiles. Data type is NUMBER.
X ordinateN, Y ordinateN	Specifies the X and Y values of a vertex (coordinate pair) in a geometry. Up to 125 pairs may be added in a single call. Data type is NUMBER.

Returns

This function returns the SDO_GID of the new geometry. Data type is NUMBER.

Usage Notes

Note: This procedure is likely to be replaced in a future release of Spatial Cartridge.

This function inserts the coordinates into the <layername>_SDOGEOM table, tessellates the geometry (creates the index), and returns a unique SDO_GID corresponding to the geometry.

You do not need special privileges to execute this function. However, the user who compiles it does need appropriate privileges to read and write into the database.

Query SDO_LEVEL from the <layername>_SDOLAYER table to pass the correct SDO_TILE_SIZE value to this function.

Related Topics

None

SDO_WINDOW.CLEAN_WINDOW

Purpose

This procedure removes (drops) the four tables created in the layer for the query window.

Syntax

SDO_WINDOW.CLEAN_WINDOW (*layername*);

Keywords and Parameters

layername	Specifies the name of the window layer that must be removed. Data type is VARCHAR2.

Usage Notes

Typically, you would build a layer once, and then build multiple windows and perform multiple queries using that layer. After finishing all queries, you can execute the `SDO_WINDOW.CLEAN_WINDOW()` procedure to remove the tables.

Related Topics

SDO_WINDOW.CLEANUP_GID

SDO_WINDOW.CLEANUP_GID

Purpose

This procedure removes the query window from the layer tables.

Syntax

SDO_WINDOW.CLEANUP_GID (*layername, SDO_GID*);

Keywords and Parameters

layername	Specifies the name of the window layer associated with the query window. Data type is VARCHAR2.
SDO_GID	Specifies the geometric object identifier of the query window. Data type is NUMBER.

Usage Notes

Typically, you would create a query layer once, and then build multiple query windows and perform multiple queries using that layer. The `SDO_WINDOW.CLEANUP_GID()` procedure removes a single query window from the layer. Use this procedure to avoid the overhead of removing and re-creating the tables repeatedly.

After finishing all queries, you can execute the `SDO_WINDOW.CLEAN_WINDOW()` procedure to remove the tables.

Related Topics

SDO_WINDOW.CLEAN_WINDOW( )

SDO_WINDOW.CREATE_WINDOW_LAYER

Purpose

This procedure creates the necessary tables that constitute a layer used for defining a query window.

Syntax

SDO_WINDOW.CREATE_WINDOW_LAYER (*layername*, *SDO_LEVEL*, *SDO_NUMTILES*, *SDO_DIMNUM1*, *SDO_LB1*, *SDO_UB1*, *SDO_TOLERANCE1*, *SDO_DIMNAME1*, *SDO_DIMNUM2*, *SDO_LB2*, *SDO_UB2*, *SDO_TOLERANCE2*, *SDO_DIMNAME2*)

Keywords and Parameters

layername	Specifies the name of the window layer to be created. The layer name is used to construct the four tables associated with the layer. Data type is VARCHAR2.
SDO_LEVEL	Specifies the number of times the layer should be tessellated during the indexing phase. Data type is INTEGER.
SDO_NUMTILES	Specifies the number of tiles to generate during indexing. Data type is INTEGER.
SDO_DIMNUM1, *SDO_DIMNUM2*	Specifies the number of the dimension, starting with 1. Data type is NUMBER.
SDO_LB1, *SDO_UB1*, *SDO_LB2*, *SDO_UB2*	Specifies the lower and upper bounds of this dimension. Data type is NUMBER.
SDO_TOLERANCE1, *SDO_TOLERANCE2*	Specifies the allowable variance of ordinate values within each dimension. Data type is NUMBER.
SDO_DIMNAME1, *SDO_DIMNAME2*	Specifies the name of the dimension. Data type is VARCHAR2.

Usage Notes

Because the <layername>_SDODIM table is initialized with the dimension and the bound information, only those queries that are in the same dimension should be queried against this layer. If you wish to issue a query with respect to a different dimension, you must create a new layer.

Related Topics

None

9 Partitioned Point Data Functions

Spatial Cartridge has undergone an architectural change, beginning with the 7.3.3 release. The emphasis on partitioned tables has been replaced by the improved spatial indexing features.

The functions described in this chapter are not required for creating or maintaining a spatial database, however, they are provided for convenience in working with legacy data in partitioned point data tables. They are used with SQL SELECT, INSERT, UPDATE, and DELETE statements to perform the following:

- Generate dimensions from bounded, hierarchical, or date data values
- Encode and decode dimensions
- Retrieve bounded, hierarchical, or date data values from dimensions

When using these functions in basic SQL statements, use the form: SDO_<function>. When using the functions inside a PL/SQL block, use a period (.) instead of the underscore.

This chapter contains descriptions of the spatial functions listed in Table 9–1.

Table 9–1 Partitioned Point Data Functions

Function	Purpose
SDO_BVALUETODIM	Creates a dimension from bounded data values.
SDO_COMPARE	Evaluates the relationship between two objects described by HHCODEs.
SDO_DATETODIM	Creates a dimension from an Oracle DATE data type.
SDO_DECODE	Extracts a single dimension from an HHCODE.

Table 9–1 Partitioned Point Data Functions

Function	Purpose
SDO_ENCODE	Creates an HHCODE by combining dimensions to describe an area or point.
SDO_TO_BVALUE	Extracts a bounded data value from a dimension.
SDO_TO_DATE	Extracts an Oracle DATE data type from a dimension.

Additional functions that support partitioned point data can be found in Chapter 5, “Administrative Procedures” and Appendix A, “Sample SQL Scripts and Tuning Tips”.

SDO_BVALUETODIM

Purpose

This function creates a dimension from a bounded value, which is a value contained in a set of values expressed as a lower boundary and an upper boundary.

Syntax

SDO_BVALUETODIM (*value, lower_boundary, upper_boundary, decimal_scale*)

Keywords and Parameters

value	Specifies the value for the particular dimension. Data type is NUMBER.
lower_boundary	Specifies the lower boundary of the dimension range. Data type is NUMBER.
upper_boundary	Specifies the upper boundary of the dimension range. Data type is NUMBER.
decimal_scale	Specifies the number of digits to the right of the decimal point. Data type is NUMBER.

Returns

This function returns a dimension. The data type is RAW.

Usage Notes

Example 9–1 shows the SDO_BVALUETODIM() function.

Example 9–1

```
SQL> INSERT INTO sourcetable1(SAMPLENAME,DATA_PT)
2> VALUES ('SAMPLE1',SDO_ENCODE(SDO_BVALUETODIM(10,-100,100,7),
3> SDO_BVALUETODIM(20,-100,100,7));
```

Related Topics

- SDO_ENCODE() function
- SDO_TO_BVALUE() function

SDO_COMPARE

Purpose

This function evaluates the relationship between an area or point described by an HHCODE and another HHCODE, or a range of HHCODEs expressed as an upper bound and lower bound.

Syntax

SDO_COMPARE (*hhcode_expression*, *{hhcode_expression* |
lower_bound_HHCODE,upper_bound_HHCODE})

Keywords and Parameters

hhcode_expression	Specifies an expression that evaluates to an HHCODE. Data type is RAW.
lower_bound_HHCODE	Specifies the lower bound HHCODE expression. Data type is RAW.
upper_bound_HHCODE	Specifies the upper bound HHCODE expression. Data type is RAW.

Returns

This function returns one of the following keywords:

- ENCLOSES
- EQUAL
- INSIDE
- OUTSIDE
- OVERLAP

The data type is VARCHAR2.

Usage Notes

Example 9–2 selects all points that fall within the given multidimensional range.

Example 9–2

```
SQL> SELECT SDO_GID FROM layer1_SDOINDEX WHERE
2> SDO_COMPARE(SDO_MAXCODE,
3> SDO_ENCODE(5,5),
4> SDO_ENCODE(25,25))='INSIDE';
```

Example 9–3 selects GIDs based on interaction between their spatial index tiles.

Example 9–3

```
SQL> SELECT SDO_GID FROM layer1_SDOINDEX A, layer2_SDOINDEX B
2> WHERE SDO_COMPARE(A.SDO_CODE,B.SDO_CODE) != 'OUTSIDE';
```

Related Topics

- SDO_GEOM.INTERACT() function
- SDO_GEOM.RELATE() function

SDO_DATETODIM

Purpose

This function creates a dimension from an Oracle DATE data type. The component number determines the level of resolution of the date in the dimension.

Syntax

SDO_DATETODIM (*date* [, *component*])

Keywords and Parameters

date	Specifies the calendar date. Data type is DATE.
component	Specifies the level of resolution. The component number values are defined as follows: 1 accurate to year 2 accurate to month 3 accurate to day 4 accurate to hour 5 accurate to minute 6 accurate to second The default value is 6. Data type is INTEGER.

Returns

This function returns a dimension. The data type is RAW.

Usage Notes

You must use a valid Oracle date format string.

Example 9–4 shows the `SDO_DATETODIM()` function.

Example 9–4

```
SQL> INSERT INTO sourcetable1(SAMPLENAME,DATA_PT)
2> VAUES('SAMPLE1',SDO_ENCODE(SDO_DATETODIM(TO_DATE('19-Jul-96'),
3> SDO_BVALUETODIM(100,-1000,1000,7)));
```

Related Topics

- SDO_ENCODE() function
- SDO_TO_DATE() function

SDO_DECODE

Purpose

This function extracts a single dimension from an HHCODE.

Syntax

SDO_DECODE (*hhcode_expression*, *dimension_number*)

Keywords and Parameters

hhcode_expression	Specifies an expression that evaluates to an HHCODE. Data type is RAW.
dimension_number	Specifies the dimension number to extract. Data type is INTEGER.

Returns

This function returns a dimension. The data type is RAW.

Usage Notes

The `SDO_DECODE()` function is called once for each dimension to be decoded.

Example 9–5 shows the `SDO_DECODE()` function.

Example 9–5

```
SQL> SELECT
2> SDO_TO_BVALUE(SDO_DECODE(DATA_PT,1),1,6),
3> SDO_TO_BVALUE(SDO_DECODE(DATA_PT,2),-100,100),
4> SDO_TO_DATE(SDO_DECODE(DATA_PT,3))
5> FROM sourcetable1 WHERE SAMPLENAME='SAMPLE1';
```

Related Topics

- SDO_TO_BVALUE() function
- SDO_TO_DATE() function

SDO_ENCODE

Purpose

This function combines dimensions to create the HHCODE that describes an area or point.

Syntax

SDO_ENCODE (*dimension1*[,*dimension2* ...])

Keywords and Parameters

dimension	Specifies an expression created by the SDO_BVALUETODIM or SDO_DATETODIM functions. Data type is RAW.

Returns

This function returns an HHCODE. The data type is RAW.

Usage Notes

Consider the following when using this function:

- When encoding dimensions, the order of the dimensions in the parameter list must be consistent for all rows within the table.
- This function can encode up to 32 dimensions.

Example 9–6 shows the `SDO_ENCODE()` function.

Example 9–6

```
SQL> INSERT INTO sourcetable1(SAMPLENAME,DATA_PT)
2> VALUES ('SAMPLE1',SDO_ENCODE(SDO_BVALUETODIM(50,-100, 100, 10),
3> SDO_BVALUETODIM(30,-100,100,10),
4> SDO_DATETODIM(TO_DATE('05-Jul-96'),3)));
```

Related Topics

- SDO_BVALUETODIM() function
- SDO_DATETODIM() function

SDO_TO_BVALUE

Purpose

This function returns the original bounded data value of a dimension.

Syntax

SDO_TO_BVALUE (*dimension*, *lower_boundary, upper_boundary*)

Keywords and Parameters

dimension	Specifies the dimension. Data type is RAW.
lower_boundary	Specifies the lower boundary of the dimension range. Data type is NUMBER.
upper_boundary	Specifies the upper boundary of the dimension range. Data type is NUMBER.

Returns

This function returns a bounded data value. The data type is NUMBER.

Usage Notes

This function returns a number that is the value for a dimension within the specified range. This is not necessarily the range for which the dimension was originally created.

Example 9–7 shows the `SDO_TO_BVALUE()` function.

Example 9–7

```
SQL> SELECT (SDO_TO_BVALUE(SDO_DECODE(DATA_PT,2),-100,100)
2> FROM sourcetable1 WHERE SAMPLENAME='SAMPLE1';
```

Related Topics

- SDO_DECODE() function
- SDO_BVALUETODIM() function

SDO_TO_DATE

Purpose

This function returns the original date value of a dimension.

Syntax

SDO_TO_DATE (*dimension*)

Keywords and Parameters

dimension	Specifies the dimension. Data type is RAW.

Returns

This function returns an Oracle DATE data type.

Usage Notes

Example 9–8 shows the `SDO_TO_DATE()` function.

Example 9–8

```
SQL> SELECT SDO_TO_DATE(SDO_DECODE(DATA_PT,3))
2> FROM sourcetable1 WHERE SAMPLENAME='SAMPLE1';
```

Related Topics

- SDO_DATETODIM() function
- SDO_DECODE() function

A Sample SQL Scripts and Tuning Tips

This appendix provides supplemental information to aid in setup, maintenance, and tuning of a spatial database. The scripts and tuning suggestions provided are intended as guidelines that can be adapted to the specific needs of your database.

A.1 Sample SQL Scripts

Spatial Cartridge provides sample SQL script files to show how to use dynamic SQL in a PL/SQL block to create layer tables for spatially indexed data or to administer and manipulate all the partitions of a partitioned spatial table. The scripts are available after installation in the ORACLE_HOME/md/admin directory.

A.1.1 Scripts for Spatial Indexing

This section describes the cr_spatial_index.sql and crlayer.sql scripts.

A.1.1.1 cr_spatial_index.sql Script

The cr_spatial_index.sql script file shows an example of updating the spatial index for a layer, and executing a commit after every 50 GIDs have been entered.

The procedures `SDO_ADMIN.POPULATE_INDEX()` and `SDO_ADMIN.POPULATE_INDEX_FIXED()` operate as a single transaction. To reduce the amount of rollback required to execute these procedures, you can write a routine similar to that in cr_spatial_index.sql. This script loops and calls `SDO_ADMIN.UPDATE_INDEX_FIXED()` for each GID, committing after every 50 GIDs.

```
-- cr_spatial_index.sql
--
-- Note: if geometries do not span more than 1 row, you can remove
-- the DISTINCT qualifier from the SELECT statement
```

```
--
declare
   cursor c1 is SELECT DISTINCT sdo_gid from POLYGON_SDOGEOM;
   gid number;
   i number;
begin
     i := 0;
     for r in c1 loop
       begin
        gid:= r.sdo_gid;
        sdo_admin.update_index_fixed('POLYGON', gid, 15, FALSE, FALSE, FALSE);
        exception when others then
          dbms_output.put_line('error for gid'||to_char(gid)||':  '||SQLERRM );
       end;
       i:=  i + 1;
       if i = 50 then
          commit;
          i:= 0;
       end if;
     end loop;
commit;
end;
/
```

When you call the `SDO_ADMIN.UPDATE_INDEX_FIXED()` procedure for a large data set, you may get a "snapshot too old" error message from the Oracle server. You can avoid this error by creating more or larger rollback segments. You can also try to increase the number of GIDs before committing the transaction.

Note: The cr_spatial_index.sql script is not available in your ORACLE_HOME/md/admin directory after installation. You must create this script yourself.

A.1.1.2 crlayer.sql Script

The crlayer.sql script file is a template used to create all the tables for a layer and populate the metadata in the <layername>_SDODIM and <layer-name>_SDOLAYER tables.

A.1.2 Scripts for Partitioned Point Data

This section describes the following scripts:

- altpart.sql
- drppart.sql
- sdogrant.sql

Although the scripts described in this section are available, the recommended approach is to use Oracle8 partitioning and spatial indexing.

A.1.2.1 altpart.sql Script

The altpart.sql script file shows how to use dynamic SQL in a PL/SQL procedure to modify all partitions of a Spatial Cartridge partitioned table.

The Spatial Cartridge data dictionary view used in this SQL script requires that a registered Spatial Cartridge partitioned table is specified. If the table is not registered, you can use the USER_TABLES view to select all the partition tables from the user's schema. To use the USER_TABLES view, use the following syntax:

```
SQL> SELECT TABLENAME FROM user_tables WHERE TABLENAME LIKE
2> '%tablename_P%';
```

A.1.2.2 drppart.sql Script

The drppart.sql script file shows how to use dynamic SQL in a PL/SQL procedure to drop (remove) all partitions of a Spatial Cartridge partitioned table. After running this procedure, you must run the `SDO_ADMIN.DROP_PARTITION_INFO()` procedure.

The Spatial Cartridge data dictionary view used in this SQL script requires that a registered Spatial Cartridge partitioned table is specified. If the table is not registered, you can use the USER_TABLES view to select all the partition tables from the user's schema. To use the USER_TABLES view, use the following syntax:

```
SQL> SELECT TABLENAME FROM user_tables WHERE TABLENAME LIKE
2> '%tablename_P%';
```

A.1.2.3 sdogrant.sql Script

The sdogrant.sql script file contains an administrative procedure, `PROPAGATE_GRANTS()`, which is used after calling the `SDO_ADMIN.PARTITION()` or `SDO_ADMIN.REPARTITION()` procedures.

This procedure must first be compiled by running the sdogrant.sql file. The `PROPAGATE_GRANTS()` procedure is only callable by the user who compiled it.

A.2 Tuning Tips

The following information can be used as a guideline for tuning a spatial database.

A.2.1 Data Modeling

Data modeling is very important when designing a spatial database. You should group geometries into layers based on the similarity of their attributes. Assume your data model uses line strings to represent both roads and rivers. The attributes of a road and the attributes of a river are different. Therefore, these geometries should be modeled in two different layers.

In practice, however, if the user of your application will always ask to see both the roads and rivers in a particular region (area of interest), then it may be appropriate to model roads and rivers in the same layer with a common set of attributes.

It is equally important to understand how the data in the various layers will be queried. If the user of your application is interested in querying the data based on a relationship between the layers, then you should index the layers with the same tiling level. For example, a query such as, "which roads cross rivers?" can achieve better performance if the roads and rivers layers are tiled at the same level.

It is not always critical to tile all your layers to the same level. You may find times when you have two layers that are optimally tiled to different levels (for example zipcode boundaries may be tiled to level 5 and and major roads may be tiled to level 7). If you want to ask the question, give me all the major roads that intersect a particular postal code boundary, it is not necessary to re-tile all the zipcode boundaries to level 7. You can move the postal code boundary of interest to a temporary layer and just re-tile that one geometry to level 7. You can then perform the query.

A.2.2 Understanding the Tiling Level

The following example explains how tiling is used in Spatial Cartridge.

Assume you want all the roads (line strings) that overlap a county boundary (polygon) in a spatial database containing 10 million roads. Ignoring Spatial Cartridge features for a moment, in purely mathematical terms, the problem translates into comparing all the line segments that make up each road, to the line segments and area of the county boundary to see if there is any intersection. This geometry-to-geometry comparison is very expensive.

Spatial Cartridge simplifies this calculation by approximating each geometry with fixed-sized tiles. The primary filter in Spatial Cartridge translates the problem to show all the roads that have a tile equal to a tile that approximates the polygon. The result of this is a superset of the final answer.

The secondary filter (a true geometry-to-geometry comparison) can now be applied to the candidates that returned from the Spatial Cartridge primary filter, instead of to every road in the database.

Picking the correct tile size for fixed tiling is one of the most important factors in attaining good performance. If the tile size you select is too small, you could end up generating thousands of tiles per geometry. Also, the process of tiling a query window (like the county boundary mentioned previously) may become very time consuming.

At the same time, you do not want to choose tiles that are too big. This would defeat the purpose of the Spatial Cartridge primary filter. If the tiles are too big, then too many geometries are returned from the primary filter and are sent to the more costly secondary filter.

Keep in mind that the tile size you choose should also depend on if the query window (area of interest) is already defined in the database. If the query window is defined in the database, (that is, if the spatial tables and spatial indexes already exist), then you should choose a smaller tile size. Assume the State layer and the Highway layer are already defined in the database. You could perform a spatial join query such as, "which interstate highways go through the state?" without incurring the overhead of tiling because the query window is already defined in the database. If, on the other hand, you are creating the query window dynamically, you have to factor in the time it takes to define and index the query window. In this case, you should choose a larger tile size to reduce the time it takes to define and index the query window.

Oracle recommends running the `SDO_TUNE.ESTIMATE_TILING_LEVEL()` function on your data set to get an initial tiling level estimate. This may not be your final answer, but it will be a good level to start your analysis. In general, it is also recommended that you take a random sample of your data and check the query performance at different levels of tiling. This would give an indication of what is the best tiling level for the total data set.

A.2.3 Database Sizing

Properly choosing rollback segments and tablespaces are important for getting good performance from Spatial Cartridge. Therefore, it is very important to read the *Oracle8 Administrator's Guide* and understand the concepts of tablespaces and rollbacks.

Here are some general guidelines to consider:

- Always make sure that you have enough rollback space to run the `SDO_ADMIN.POPULATE_INDEX_FIXED()` procedure. To reduce rollback segment size, run the `SDO_ADMIN.UPDATE_INDEX_FIXED()` procedure in a loop (see Section A.1.1.1).

- Create separate tablespaces for Data Layers, Indexes, and Rollback Segments.
- Properly define initial extents, next extents, and pctincrease for data layer tables.
- Define the initial extent as small as possible when you create the <layername>_SDOLAYER and the <layername>_SDODIM tables. These tables contain a few rows each and a small initial extent will reduce the amount of wasted space.
- Use the `SDO_ADMIN.VALIDATE_GEOMETRY()` procedure to ensure correctness of geometries in the data sets. Entering incorrect data may lead to unexpected behavior in index creation and in the `SDO_GEOM.RELATE()` functions.
- Always build a B-tree index on the SDO_GID column of the <layername>_SDOGEOM table before attempting to call the `SDO_ADMIN.POPULATE_INDEX_FIXED()`, `SDO_ADMIN.UPDATE_INDEX_FIXED()`, `SDO_ADMIN.POPULATE_INDEX()`, or `SDO_ADMIN.UPDATE_INDEX()` procedure.
- For fixed-size tiling, always build a B-tree index on the SDO_CODE column of the <layername>_SDOINDEX table before trying any queries using this table.
- Always build a B-tree index on the SDO_GID column of the <layername>_SDOINDEX table if individual SDO_GIDs will be used as query windows for other Spatial Cartridge layers.
- Visualizing the indexing tiles, as described in Section A.2.8, can lead to a greater understanding of the tuning process with respect to the size of the tiles.
- For variable-sized tiling, always build a B-tree index on the SDO_GROUPCODE column of the <layername>_SDOINDEX table before trying any queries using this table.

A.2.4 Tuning Point Data

Point data, unlike line and polygon data, has the unique characteristic of containing one tile per point. This section describes how to improve the performance of queries on point data.

A.2.4.1 Efficient Queries for Point Data

When querying point data with a rectangular query window, you can take advantage of the nature of these geometries to improve performance.

A rectangle can be defined by its lower-left and upper-right coordinates (Xmin, Ymin and Xmax, Ymax). A point has a single set of coordinates (Px, Py). When your area-of-interest is a rectangle, instead of using the `SDO_GEOM.RELATE()`function in the secondary filter, you can use simple SQL comparison operators as follows:

```
SELECT sdo_gid, sdo_x1, sdo_y1
FROM  cities_sdogeom,
      (SELECT a.sdo_gid gid1
      FROM cities_sdoindex a,
           window_sdoindex b
      WHERE b.sdo_gid = [area of interest id]
        AND a.sdo_code = b.sdo_code)
           WHERE sdo_gid = gid1
            AND sdo_x1 BETWEEN Xmin AND Xmax
            AND sdo_y1 BETWEEN Ymin AND Ymax;
```

The DISTINCT clause is not necessary in the primary filter of the query because a point contains only a single tile in the spatial index.

A.2.4.2 Efficient Schema for Point Layers

Because a point is always referenced by only one tile in a spatial index, for additional performance, you can place the columns normally found in the <layername>_SDOINDEX table in the <layername>_SDOGEOM table. This will save you the cost of joining the <layername>_SDOINDEX and <layername>_SDOGEOM tables.

You still need to create an updatable view for the <layername>_SDOINDEX table that selects the appropriate columns from the <layername>_SDOGEOM table. This is because functions such as `SDO_ADMIN.UPDATE_INDEX_FIXED()` and `SDO_ADMIN.POPULATE_INDEX_FIXED()` expect a <layername>_SDOINDEX table to exist. Create the view using "instead of" triggers for insert, delete, and update such that the appropriate columns in the <layername>_SDO_GEOM table are updated. The following example shows how to use "instead of" triggers:

```
CREATE OR REPLACE TRIGGER mytrig INSTEAD OF INSERT ON points_sdoindex
     REFERENCING new AS n
     FOR EACH ROW
     BEGIN
          UPDATE points_sdogeom SET points_sdogeom.sdo_code = :n.sdo_gid;
     END;
CREATE OR REPLACE TRIGGER mydeltrig INSTEAD OF DELETE ON points_sdoindex
    REFERENCING old AS n
    FOR EACH ROW
    BEGIN
          UPDATE points_sdogeom SET points_sdogeom.sdo_code = NULL
          WHERE  points_sdogeom.sdo_gid = :n.sdo_gid;
    END;
```

The following example shows a window query of a layer containing point data when the window layer contains one rectangle:

```
SELECT sdo_gid, sdo_x1, sdo_y1
FROM  points_sdogeom a,
      window_sdoindex b
WHERE b.sdo_gid = [area of interest id]
  AND a.sdo_code = b.sdo_code)
  AND sdo_x1 BETWEEN Xmin AND Xmax
  AND sdo_y1 BETWEEN Ymin AND Ymax;
```

A.2.5 Tuning Spatial Join Queries

There are some helpful hints you can place in your spatial join queries to improve performance. The remainder of this section describes some of the hints you can use. For more information on hints, see the *Oracle8 Tuning* manual.

A.2.5.1 Using the NO_MERGE, INDEX, and USE_NL Hints

A spatial join takes place between two layers. When the two layers being joined are line or polygon layers, the spatial join query contains two DISTINCT clauses: one in the inner SELECT clause and the other in the outer SELECT clause. The Oracle optimizer ignores the inner DISTINCT clause to save on the cost of sorting. However, if the inner DISTINCT clause is ignored, the secondary filter gets called many more times than it needs to be. This can have a significant impact on performance because the secondary filter is an expensive operation. Use the NO_MERGE hint to prevent the optimizer from ignoring the inner DISTINCT clause.

In a spatial join, all the tiles from one layer are compared to all the tiles from another layer. The Oracle server performs a full table scan on one <layer-name>_SDOINDEX table, (preferably the smaller of the two), and an index lookup on the other <layername>_SDOINDEX table. Use the INDEX and USE_NL hints to force the optimizer to perform the full table scan on the smaller of the two <layer-name>_SDOINDEX tables being compared.

The following example shows a spatial join between line (road) and polygon (county) data. The query answers the question, "which counties intersect major roads?"

```
SELECT /*+ cost
           ordered use_nl(COUNTY_sdogeom)
           index (COUNTY_sdogeom NAME_OF_SDO_GID_INDEX)
       */
       COUNTY_sdogeom.SDO_GID,
       COUNTY_sdogeom.SDO_ESEQ,
       COUNTY_sdogeom.SDO_SEQ,
       COUNTY_sdogeom.SDO_X1,COUNTY_sdogeom.SDO_Y1,
       COUNTY_sdogeom.SDO_X2,COUNTY_sdogeom.SDO_Y2,
       COUNTY_sdogeom.SDO_X3,COUNTY_sdogeom.SDO_Y3,
       COUNTY_sdogeom.SDO_X4,COUNTY_sdogeom.SDO_Y4,
       COUNTY_sdogeom.SDO_X5,COUNTY_sdogeom.SDO_Y5,
       COUNTY_sdogeom.SDO_X6,COUNTY_sdogeom.SDO_Y6,
       COUNTY_sdogeom.SDO_X7,COUNTY_sdogeom.SDO_Y7,
       COUNTY_sdogeom.SDO_X8,COUNTY_sdogeom.SDO_Y8
FROM (SELECT DISTINCT gid_a gid1
      FROM (SELECT /*+ index (a NAME_OF_SDO_CODE_INDEX)
                       index (b NAME_OF_SDO_CODE_INDEX)
                       use_nl (a b)
                       no_merge */
            DISTINCT a.sdo_gid gid_a,
                     b.sdo_gid gid_b
            FROM COUNTY_SDOINDEX a,
                 MAJOR_ROAD_SDOINDEX b
            WHERE a.sdo_code = b.sdo_code)
      WHERE sdo_geom.relate('COUNTY', gid_a, 'ANYINTERACT',
                            'MAJOR_ROAD',gid_b) <> 'FALSE'),
      COUNTY_sdogeom
WHERE COUNTY_sdogeom.sdo_gid = gid1;
```

A.2.5.2 Spatial Join Queries with Point Layers

The following example shows a spatial join between line (road) and point (street address) data. The query answers the question, "which addresses are on a major road?"

```
SELECT /*+ cost
           ordered use_nl (STREET_ADDRESS_sdogeom)
           index (STREET_ADDRESS_sdogeom NAME_OF_SDO_GID_INDEX)
       */
       STREET_ADDRESS_sdogeom.SDO_GID,
       STREET_ADDRESS_sdogeom.SDO_X1,
       STREET_ADDRESS_sdogeom.SDO_Y1
FROM (SELECT DISTINCT gid_a gid1
      FROM (SELECT /*+ index (a NAME_OF_SDO_CODE_INDEX)
                       index (b NAME_OF_SDO_CODE_INDEX)
                       use_nl (a b) */
                     a.sdo_gid gid_a,
                     b.sdo_gid gid_b
            FROM STREET_ADDRESS_SDOINDEX a,
                 MAJOR_ROAD_SDOINDEX b
            WHERE a.sdo_code = b.sdo_code)
      WHERE sdo_geom.relate('STREET_ADDRESS', gid_a, 'ANYINTERACT',
                            'MAJOR_ROAD',gid_b) <> 'FALSE'),
      COUNTY_sdogeom
WHERE COUNTY_sdogeom.sdo_gid = gid1;
```

The inner DISTINCT clause is not necessary for spatial joins where one of the layers contains point data. Therefore, the NO_MERGE hint is not necessary. This is because points contain only one tile in the spatial index.

The following example shows a spatial join between polygon (county) and point (street address) data. The query generates a report that displays how many addresses are associated with each county.

If you can assume that each street address is associated with a single county, you can significantly speed up this query. Because points contain only a single tile in the spatial index, any street address tile that matches only one county tile in the primary filter does not need to go through the expensive secondary filter.

```
SELECT county_gid, count(street_gid)
FROM  (SELECT poly.sdo_gid county_gid, street.sdo_gid street_gid
       FROM  STREET_ADDRESS_sdoindex street,
             (SELECT sdo_code county_sdo_code,
                     count(sdo_gid) interacts
              FROM CENSUS_COUNTY_sdoindex
              GROUP by sdo_code
             ) counts,
             CENSUS_COUNTY_sdoindex  poly
       WHERE street.sdo_code = counts.county_sdo_code
         AND poly.sdo_code   = street.sdo_code
         AND (counts.interacts = 1
              OR
              sdo_geom.relate('STREET_ADDRESS', street.sdo_gid,
                              'ANYINTERACT',
                              'CENSUS_COUNTY',poly.sdo_gid) <> 'FALSE'
             )
      )
GROUP BY county_gid;
```

A.2.6 Using Customized Geometry Types

Spatial Cartridge supports three geometry types: points, lines, and polygons. If your data contains another type, such as a circle or arc, then you must choose the supported type that best approximates your desired type. For example, a circle can be defined as a multi-sided polygon. Obviously, the more coordinates in the element, the better the approximation will be.

Although customized types are not supported, you do not have to lose your knowledge of the type. After storing the approximated element, create another element in that geometry with ETYPE=0. Spatial Cartridge ignores elements of ETYPE=0. You can then write your own routines to handle your specialized geometry type.

A.2.7 Performing Secondary Filter Queries and the Redo Log

When the Oracle server processes SQL statements that require sorting, such as statements containing an ORDER BY or DISTINCT clause, the Oracle server stores the result set in a temporary storage area. The result set is then sorted. If the SORT_AREA_SIZE is insufficient for holding the result set in memory, then some data may be written to disk and an entry is written in the redo log.

The RELATE() and INTERACT() secondary filters issue SQL statements internally that contain DISTINCT and ORDER BY clauses. If the SORT_AREA_SIZE initialization parameter is too small for processing the secondary filters, then some sorting may occur on disk, which causes entries to be written in the redo log. This may affect performance. For better performance, increase the SORT_AREA_SIZE parameter to force sorting to occur in memory.

A.2.8 Visualizing the Spatial Index (Drawing Tiles)

To select an appropriate tiling level, it may help to visualize the tiles covering your geometries. Through visualization, you can determine how many tiles are used for each object, the size of the tiles, and how well the edges of your geometry are covered. The basic algorithm is:

1. Select the edges of the tiles represented by the index entries.
2. Plot the tiles on a two-dimensional grid.
3. Plot your geometries on the same grid.

The Spatial Cartridge spatial index is represented internally as a linear quadtree. The structure used to represent the linear quadtree is composed of two components: a data component and a metadata component. The data component of the linear quadtree is stored in the SDO_CODE column, and the metadata component is stored in the SDO_META column.

The SDO_META column is not required for spatial queries. However, by combining the SDO_META column with the SDO_CODE column, the tiles of any geometry or of the entire data set can be decoded. This capability allows the tiles to be visualized.

Two Spatial Cartridge internal functions have been made visible in order to describe the tiles. These functions were part of a previous release of Oracle Spatial Data Option, and are currently reserved for internal use only. The functions are not recommended for general use, except for this visualization example. Use the following syntax for the internal functions:

```
hhcellbndry (sdo_code || sdo_meta, sdo_dimnum, sdo_lb, sdo_ub,
    hhlength(sdo_code || sdo_meta) {'MIN' | 'MAX'})
```

In the following examples, the dimension boundaries were assumed to be -180 to 180, and -90 and 90. The dimensional information is stored in the <layer-name>_SDODIM table.

If you used `SDO_ADMIN.UPDATE_INDEX_FIXED()` or `SDO_ADMIN.POPULATE_INDEX_FIXED()` to generate your spatial index, replace "sdo_code || sdo_meta" with sdo_tile in the SQL statements that follow.

The following SQL query can be used to decode all the index entries in a <layername>_SDOINDEX table. The example returns the coordinates of the lower-left and upper-right corners of each tile.

```
SELECT hhcellbndry (sdo_code || sdo_meta, 1, -180.000000000, 180.000000000,
           hhlength (sdo_code || sdo_meta), 'MIN') min_x,
       hhcellbndry (sdo_code || sdo_meta, 1, -180.000000000, 180.000000000,
           hhlength (sdo_code || sdo_meta), 'MAX') max_x,
       hhcellbndry (sdo_code || sdo_meta, 2, -90.000000000, 90.000000000,
           hhlength (sdo_code || sdo_meta), 'MIN') min_y,
       hhcellbndry (sdo_code || sdo_meta, 2, -90.000000000, 90.000000000,
           hhlength (sdo_code || sdo_meta), 'MAX') max_y
FROM (SELECT DISTINCT sdo_code, sdo_meta FROM <layername>_sdoindex);
```

The following SQL query can be used to decode the index entries for a specific geometry stored in a <layername>_SDOINDEX table:

```
SELECT hhcellbndry (sdo_code || sdo_meta, 1, -180.000000000, 180.000000000,
          hhlength (sdo_code || sdo_meta), 'MIN') min_x,
       hhcellbndry (sdo_code || sdo_meta, 1, -180.000000000, 180.000000000,
          hhlength (sdo_code || sdo_meta), 'MAX') max_x,
       hhcellbndry (sdo_code || sdo_meta, 2, -90.000000000, 90.000000000,
          hhlength (sdo_code || sdo_meta), 'MIN') min_y,
       hhcellbndry (sdo_code || sdo_meta, 2, -90.000000000, 90.000000000,
          hhlength (sdo_code || sdo_meta), 'MAX') max_y
FROM <layername>_sdoindex
WHERE sdo_gid = <geometry id>;
```

B
Data Dictionary

The Spatial Cartridge data dictionary is a set of tables owned by the database user mdsys. An extension to the Oracle8 data dictionary, it automatically maintains information about spatial tables, columns, and partitions. The Spatial Cartridge data dictionary is created during the installation process. All nonspatial attribute information is maintained in the Oracle8 data dictionary.

The Spatial Cartridge data dictionary has public views that provide extensive information about spatial tables. This appendix contains descriptions of the views that are available.

Note: Only the partitioned point routines use the Spatial Cartridge data dictionary.

The following views are publicly available:

- ALL_MD_COLUMNS
- ALL_MD_DIMENSIONS
- ALL_MD_EXCEPTIONS
- ALL_MD_LOADER_ERRORS
- ALL_MD_PARTITIONS
- ALL_MD_TABLES
- ALL_MD_TABLESPACES
- DBA_MD_COLUMNS

- DBA_MD_DIMENSIONS
- DBA_MD_EXCEPTIONS
- DBA_MD_LOADER_ERRORS
- DBA_MD_PARTITIONS
- DBA_MD_TABLES
- DBA_MD_TABLESPACES
- USER_MD_COLUMNS
- USER_MD_DIMENSIONS
- USER_MD_EXCEPTIONS
- USER_MD_LOADER_ERRORS
- USER_MD_PARTITIONS
- USER_MD_TABLES
- USER_MD_TABLESPACES

WARNING: Do not delete or modify any of the tables in the mdsys account. This corrupts the Spatial Cartridge data dictionary.

ALL_MD_COLUMNS

Returns a list of all columns that are part of spatial tables.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	name of the spatial table
COLUMN_NAME	name of the column
DATA_TYPE	data type of the column
DATA_LENGTH	length of the column in bytes
DATA_PRECISION	scale for NUMBER data type, binary precision for FLOAT data type, and NULL for all other data types
DATA_SCALE	digits to right of decimal point in an HHCODE or a number

Column	Description
NDIM	number of dimensions in the HHCODE column (It is NULL for all other data types.)
MAX_LEVEL	maximum number of levels in the column
NULLABLE	indicates if column allows NULL values
PARTITION_KEY	indicates if column is the partition key column; only one is allowed per partitioned table
COLUMN_ID	sequence number of the column as created
DEFAULT_LENGTH	length of the default value for the column
NUM_DISTINCT	number of distinct values in each column of the table
LOW_VALUE	lowest value for tables with three or fewer rows (It is the second-lowest value in the column for tables with more than three rows.)
HIGH_VALUE	highest value for tables with three or fewer rows (It is the second-highest value in the column for tables with more than three rows.)

ALL_MD_DIMENSIONS

Returns a list of all dimensions that are part of HHCODE columns.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	name of the spatial table
COLUMN_NAME	name of the column
DIMENSION_NAME	name of the dimension
DIMENSION_NUMBER	dimension number
LOWER_BOUND	lower boundary of the dimension range
UPPER_BOUND	upper boundary of the dimension range
SCALE	scale of the dimension
RECURSION_LEVEL	number of levels encoded in the HHCODE

ALL_MD_EXCEPTIONS

Contains information about spatial tables that should be removed (dropped) as a result of some failed operation, such as a failed load.

Column	Description
OWNER	owner of the object
NAME	object name
OPERATION	operation during which the failure occurred
CCHH	common code HHCODE

ALL_MD_LOADER_ERRORS

Contains the current status of a file that was loaded into a table using SD*Loader.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	spatial table name
FILENAME	SLF file name
ROWS_LOADED	number of rows loaded before failure

ALL_MD_PARTITIONS

Returns a list of all the partitioned tables that are part of a user-accessible spatial table.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	name of the spatial table
PARTITION_TABLE_NAME	name of the partitioned table
CLASS	class of partition: NODE or LEAF
COMMON_LEVEL	number of levels of resolution of the common HHCODE for the partition
COMMON_HHCODE	common HHCODE substring for the partition
OFFLINE_STATUS	status of partition: ONLINE or OFFLINE

Column	Description
ARCHIVE_DATE	date of last archive

ALL_MD_TABLES

Returns a list of all the user-accessible spatial tables.

Column	Description
OWNER	owner of the table
MD_TABLE_NAME	name of the spatial table
CLASS	class of table: PARTITIONED or NON-PARTITIONED
PTAB_SEQ	number of last partitioned table created
HIGH_WATER_MARK	maximum number of rows that can be inserted into a partitioned table
OFFLINE_PATH	complete path name to directory where the table is archived
COUNT_MODE	count mode for estimating number of rows in a partition: ESTIMATE or EXACT

ALL_MD_TABLESPACES

Returns a list of all tablespaces used by spatial tables.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	name of the spatial table
TABLESPACE_NAME	name of tablespace
SEQUENCE	sequence number
STATUS	status of tablespace: ACTIVE or INACTIVE

DBA_MD_COLUMNS

Returns a list of all columns that are part of Spatial Cartridge tables.

Column	Description
OWNER	owner of the object

Column	Description
MD_TABLE_NAME	name of the spatial table
COLUMN_NAME	name of the column
DATA_TYPE	data type of the column
DATA_LENGTH	length of the column in bytes
DATA_PRECISION	scale for NUMBER data type, binary precision for FLOAT data type, and NULL for all other data types
DATA_SCALE	digits to right of decimal point in an HHCODE or a number
NDIM	number of dimensions in the HHCODE column (It is NULL for all other data types.)
MAX_LEVEL	maximum number of levels in the column
NULLABLE	indicates if column allows NULL values
PARTITION_KEY	indicates if column is the partition key column; only one is allowed per partitioned table
COLUMN_ID	sequence number of the column as created
DEFAULT_LENGTH	length of the default value for the column
NUM_DISTINCT	number of distinct values in each column of the table
LOW_VALUE	lowest value for tables with three or fewer rows (It is the second-lowest value in the column for tables with more than three rows.)
HIGH_VALUE	highest value for tables with three or fewer rows (It is the second-highest value in the column for tables with more than three rows.)

DBA_MD_DIMENSIONS

Returns a list of all dimensions that are a part of spatial tables.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	name of the spatial table
COLUMN_NAME	name of the column
DIMENSION_NAME	name of the dimension

Column	Description
DIMENSION_NUMBER	dimension number
LOWER_BOUND	lower boundary of the dimension range
UPPER_BOUND	upper boundary of the dimension range
SCALE	scale of the dimension
RECURSION_LEVEL	number of levels encoded in the HHCODE

DBA_MD_EXCEPTIONS

Contains information about spatial tables that should be removed (dropped) as a result of some failed operation, such as a failed load.

Column	Description
OWNER	owner of the object
NAME	object name
OPERATION	operation during which the failure occurred
CCHH	common code HHCODE

DBA_MD_LOADER_ERRORS

Contains the current status of a file that was loaded into a table using SD*Loader.

Column	Description
OWNER	owner of the table where the error occurred
MD_TABLE_NAME	spatial table name
FILENAME	SLF file name
ROWS_LOADED	number of rows loaded before failure

DBA_MD_PARTITIONS

Returns a list of all the partitioned tables.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	name of the spatial table

Column	Description
PARTITION_TABLE_NAME	name of the partitioned table
CLASS	class of partition: NODE or LEAF
COMMON_LEVEL	number of levels of resolution of the common HHCODE for the partition
COMMON_HHCODE	common HHCODE substring for the partition
OFFLINE_STATUS	status of partition: ONLINE or OFFLINE
ARCHIVE_DATE	date of last archive

DBA_MD_TABLES

Returns a list of all the spatial tables.

Column	Description
OWNER	owner of the table
MD_TABLE_NAME	name of the spatial table
CLASS	class of table: PARTITIONED or NON-PARTITIONED
PTAB_SEQ	number of last partitioned table created
HIGH_WATER_MARK	maximum number of rows that can be inserted into a partitioned table
OFFLINE_PATH	complete path name to directory where the table is archived
COUNT_MODE	count mode for estimating number of rows in a partition: ESTIMATE or EXACT

DBA_MD_TABLESPACES

Returns a list of all tablespaces used by spatial tables.

Column	Description
OWNER	owner of the object
MD_TABLE_NAME	name of the spatial table
TABLESPACE_NAME	name of tablespace
SEQUENCE	sequence number
STATUS	status of tablespace: ACTIVE or INACTIVE

USER_MD_COLUMNS

Returns a list of all the HHCODE columns that are part of tables owned by the user.

Column	Description
MD_TABLE_NAME	name of the spatial table
COLUMN_NAME	name of the spatial table
DATA_TYPE	data type of the column
DATA_LENGTH	length of the column in bytes
DATA_PRECISION	scale for NUMBER data type, binary precision for FLOAT data type, and NULL for all other data types
DATA_SCALE	digits to right of the decimal point in an HHCODE or a number
NDIM	number of dimensions in the HHCODE column (It is NULL for all other data types.)
MAX_LEVEL	maximum number of levels in the column
NULLABLE	indicates if column allows NULL values
PARTITION_KEY	indicates if column is the partition key column; only one allowed per partitioned table
COLUMN_ID	sequence number of the column as created
DEFAULT_LENGTH	length of the default value for the column
NUM_DISTINCT	number of distinct values in each column of the table
LOW_VALUE	lowest value for tables with three or fewer rows (It is the second-lowest value in the column for tables with more than three rows.)
HIGH_VALUE	highest value for tables with three or fewer rows (It is the second-highest value in the column for tables with more than three rows.)

USER_MD_DIMENSIONS

Returns a list of all dimensions that are part of HHCODE columns owned by the user.

Column	Description
MD_TABLE_NAME	name of the spatial table

Column	Description
COLUMN_NAME	name of the column
DIMENSION_NAME	name of the dimension
DIMENSION_NUMBER	dimension number
LOWER_BOUND	lower boundary of dimension range
UPPER_BOUND	upper boundary of dimension range
SCALE	scale of the dimension
RECURSION_LEVEL	number of levels encoded in the HHCODE

USER_MD_EXCEPTIONS

Contains information about spatial tables that should be removed (dropped) as a result of some failed operation, such as a failed load.

Column	Description
NAME	object name
OPERATION	operation during which the failure occurred
CCHH	common code HHCODE

USER_MD_LOADER_ERRORS

Contains the current status of a file that was loaded into a table using SD*Loader.

Column	Description
MD_TABLE_NAME	spatial table name
FILENAME	SLF file name
ROWS_LOADED	number of rows loaded before failure

USER_MD_PARTITIONS

Returns a list of all the partitioned tables that are part of spatial tables owned by the user.

Column	Description
MD_TABLE_NAME	name of the spatial table

Column	Description
PARTITION_TABLE_NAME	name of the partition
CLASS	class of partition: NODE or LEAF
COMMON_LEVEL	number of levels of resolution of the common HHCODE for the partition
COMMON_HHCODE	common HHCODE substring for the partition
OFFLINE_STATUS	status of partition: ONLINE or OFFLINE
ARCHIVE_DATE	date of last archive

USER_MD_TABLES

Returns a list of all the spatial tables owned by the user.

Column	Description
MD_TABLE_NAME	name of the spatial table
CLASS	class of table: PARTITIONED or NON-PARTITIONED
PTAB_SEQ	number of last sequence created
HIGH_WATER_MARK	maximum number of rows that can be inserted into a partitioned table
OFFLINE_PATH	complete path name to directory where the table is archived
COUNT_MODE	count mode for estimating number of rows in a partition: ESTIMATE or EXACT

USER_MD_TABLESPACES

Returns a list of all tablespaces used by spatial tables.

Column	Description
MD_TABLE_NAME	name of the spatial table
TABLESPACE_NAME	name of tablespace
SEQUENCE	sequence number
STATUS	status of the tablespace: ACTIVE or INACTIVE

C
Messages and Codes

MDSQL-00001: partition is OFFLINE

Cause: An MDSQL operation was attempted on a partition that is OFFLINE.

Action: Restore the partition and try the operation again.

MDSQL-00002: PK is out of bounds

Cause: The partition key for the record being inserted belongs in another partition.

Action: Insert the record into the correct partition. The correct partition can be identified using the GET_PARTITION_NAME() function.

MDSQL-00003: updates that move the PK are not supported

Cause: The update of the partition key would result in the record belonging to another partition.

Action: Use the MD_DML.MOVE_RECORD() procedure to update the partition key and move the record to the correct partition.

Additional error messages are documented in the *Oracle8 Error Messages* manual in the range of 13000 to 13199.

Glossary

area

An extent or region of dimensional space.

attribute

Descriptive information characterizing a geographical feature such as a point, line, or area.

attribute data

Nondimensional data that provides additional descriptive information about multidimensional data, for example a class or feature such as a bridge or a road.

boundary

1. The lower or upper extent of the range of a dimension, expressed by a numeric value.
2. The line representing the outline of a polygon.

Cartesian coordinate system

A coordinate system in which the location of a point in n-dimensional space is defined by distances from the point to the reference plane. Distances are measured parallel to the planes intersecting a given reference plane.

contain

To describe a geometric relationship where one object encompasses another and the inner object does not touch any boundaries of the outer. The outer object *contains* the inner object. *See also* inside.

coordinate

A set of values uniquely defining a point in an n-dimensional coordinate system.

coordinate system

A reference system for the unique definition for the location of a point in n-dimensional space.

cover

To describe a geometric relationship in which one object encompasses another and the inner object touches the boundary of the outer object in one or more places.

data dictionary

A repository of information about data. A data dictionary stores relational information on all the objects in a database.

decompose

To separate or resolve into constituent parts or elements, or into simpler compounds.

dimensional data

Data that has one or more dimensional components and is described by multiple values.

disjoint

A geometric relationship where two objects do not interact in any way. Two *disjoint* objects do not share any element or piece of their geometry.

equal

A geometric relationship in which two objects are considered to represent the same geometric figure. The two objects must be composed of the same number of points, however, the ordering of the points defining the two objects' geometries may differ (clockwise or counter-clockwise).

extent

A rectangle bounding a map, the size of which is determined by the minimum and maximum map coordinates.

feature

An object in a spatial database with a distinct set of characteristics.

geographically referenced data

See spatiotemporal data.

georeferenced data

See spatiotemporal data.

geographical information system

A computerized database management system used for the capture, conversion, storage, retrieval, analysis, and display of spatial data.

GIS

See geographical information system.

grid

A data structure composed of points located at the nodes of an imaginary grid. The spacing of the nodes is constant in both the horizontal and vertical directions.

HHCODE

A data type representing the intersection point of multiple dimensions. It encodes these multiple dimensions into a unique, linear value. The HHCODEs are used for both spatial indexing and partitioned point data.

high-water mark

Expressed in number of records and associated with Spatial Cartridge partitioned table structure, it defines the maximum number of records to store in a table before decomposing another level. The high-water mark determines the maximum size of a partition within the Spatial Cartridge table. Partitioned tables are an alternative to spatial indexing.

hole

A polygon can include subelements that negate sections of its interior. For example, consider a polygon representing a map of a buildable land with an inner polygon (a hole) representing where a lake is located.

homogeneous

Spatial data of one feature type such as points, lines, or regions.

hyperspatial data

In mathematics, any space comprising more than the three standard x, y, and z dimensions, also referred to as multidimensional data.

index

Identifier that is not part of a database and used to access stored information.

inside

To describe a geometric relationship where one object is surrounded by a larger object and the inner object does not touch the boundary of the outer. The smaller object is *inside* the larger. *See also* contain.

key

A field in a database used to obtain access to stored information.

keyword

Synonym for reserved word.

latitude

North / South position of a point on the Earth defined as the angle between the normal to the Earth's surface at that point and the plane of the equator.

line

A geometric object represented by a series of points, or inferred as existing between two coordinate points.

longitude

East / West position of a point on the Earth defined as the angle between the plane of a reference meridian and the plane of a meridian passing through an arbitrary point.

multidimensional data

See hyperspatial data.

partition

1. The spatial table that contains data only for a unique bounded n-dimensional space.
2. The process of grouping data into partitions that maintain the dimensional organization of the data.

partition key column

The primary HHCODE column that is used to dimensionally partition the data. One HHCODE data type column must be identified as the partition key for the table to be registered as partitionable in the Spatial Cartridge data dictionary. There can only be one partition key per spatial table. Note that this is only used for partitioned point data, and not spatially indexed data.

partitioned table

The spatial logical table structure that contains one or more partitions. Use partitioned tables only if you are dealing with a very large amount of point data (over 50 GB).

polygon

A class of spatial objects having a nonzero area and perimeter, and representing a closed boundary region of uniform characteristics.

proximity

A measure of inter-object distance.

query

A set of conditions or questions that form the basis for the retrieval of information from a database.

query window

Area within which the retrieval of spatial information and related attributes is performed.

RDBMS

See Relational Database Management System.

recursion

A process, function, or routine that executes continuously until a specified condition is met.

region

An extent or area of multidimensional space.

Relational Database Management System (RDBMS)

A computer program designed to store and retrieve shared data. In a relational system, data is stored in tables consisting of one or more rows, each containing the same set of columns. Oracle8 is a relational database management system. Other types of database systems are called hierarchical or network database systems.

resolution

The number of subdivision levels of data.

scale

1. The number of digits to the right of the decimal point in a number representing the level of resolution of an HHCODE.
2. The ratio of the distance on a map, photograph, or image to the corresponding image on the ground, all expressed in the same units.

SD*Converter

A utility used with previous versions of Spatial Data Option to prepare data for loading into spatial tables. Loading is now accomplished through SQL*Loader.

SLF

See Spatial Load Format.

sort

The operation of arranging a set of items according to a key that determines the sequence and precedence of items.

spatial

A generic term used to reference the mathematical concept of n-dimensional data.

spatial data

Data that is referenced by its location in n-dimensional space. The position of spatial data is described by multiple values. *See also* hyperspatial data.

spatial database

A database containing information indexed by location.

spatial data model

A model of how objects are located on a spatial context.

Spatial Cartridge data dictionary

An extension of the Oracle8 data dictionary. It keeps track of the number of partitions created in a spatial table. The Spatial Cartridge data dictionary is owned by user mdsys. The data dictionary is used only by the partitioned point routines.

spatial data structures

A class of data structures designed to store spatial information and facilitate its manipulation.

Spatial Load Format (SLF)

The format used to load data into spatial tables in a previous version of Spatial Data Option.   Loading is now accomplished with the standard SQL*Loader.

spatial query

A query that includes criteria for which selected features must meet location conditions.

spatiotemporal data

Data that contains time and/or location components as one of its dimensions, also referred to as geographically referenced data or georeferenced data.

SQL*Loader

A utility to load formatted data into spatial tables.

tessellation

The process of covering a geometry with rectangular tiles without gaps or overlaps.

tiling

See tessellation.

touch

To describe a geometric relationship where two objects share a common point on their boundaries, but their interiors do not intersect.

Index