

Oracle Spatial

*An Oracle Technical White Paper
May 2001*

ABSTRACT	1
1.0 INTRODUCTION.....	1
2.0 SPATIAL DATA AND ORDBMS.....	3
2.1 Object-Relational Database Management Systems (ORDBMS).....	3
2.2 Challenges of Spatial Databases.....	3
2.2.1 Geometry	3
2.2.2 Distribution of Objects in Space.....	4
2.2.3 Temporal Changes	4
2.2.4 Data Volume.....	4
2.3 Requirements of a Spatial Database System.....	4
2.3.1 Classification of Space.....	5
2.3.2 Data Model	5
2.3.3 Query Language	5
2.3.4 Spatial Query Processing	6
2.3.5 Spatial Indexing.....	6
3.0 ORACLE SPATIAL.....	7
3.1 Spatial Data Modeling.....	7
3.2 Operations on the Spatial types	11
3.3 Spatial Indexing.....	13
3.4 Query Processing	14
4.0 DIFFERENCES BETWEEN SPATIAL RELEASE 8 <i>i</i> AND 9 <i>i</i>	16
4.1 Spatial Aggregates	17
4.2 Function-Based Index Support.....	17
4.3 Geodetic Coordinate Support.....	18
4.4 Partitioning Support for Spatial Indexes.....	18
4.5 Linear Referencing Support.....	18
4.6 Performance Enhancements	19
5.0 SUMMARY	19

ABSTRACT

Object-relational databases have become the new standard for addressing the growing data management and analysis needs of non-traditional database applications such as Multimedia and Spatial Information Systems. Two critical issues must be resolved in order to effectively meet the requirements of these applications. They are: (i) representation and (ii) content-based search of multimedia and spatial data. Oracle Spatial addresses these issues for spatial data by providing an object data type (SDO_GEOMETRY), indexing capability, and functions/operators on SDO_GEOMETRY. It enables spatial data to be stored, accessed, and analyzed quickly and efficiently in an Oracle9i database. This gives application developers the facility to store all location (geographically referenced) information within an industry standard database server without having to resort to custom-built external indexes and functions to get the functionality they need. Users of spatial data have access to standard Oracle9i features, plus enhanced features such as bigger database size limits, faster backup and recovery, and Java stored procedures in the database.

1.0 INTRODUCTION

Extensible database management systems aim to make data management easier and more natural to users or applications such as web-content management, location-based services, urban planning, utilities, transportation, and remote sensing. Databases are traditionally used in business and administrative applications. In such applications, the common data types encountered are integer, float, character, monetary-unit, and date. The type of operations performed on these data types are simple arithmetic operations like addition, subtraction, and sorting. This limited set of data types and operations makes the modeling of real-world spatial applications extremely difficult. Hence, the recent advances in commercial database systems have focused on efficiently storing and managing complex information like spatial data. We discuss how these new object-relational and extensible databases can be used to solve the problems posed by spatial information management.

A simple example of spatial (or location) data is a street address. Geocoding (a process of converting an address to a longitude/latitude coordinate pair) results in a location which can then be used to determine the spatial relationships

among street addresses or between a street address and some linear spatial feature, such as a road, or an area feature, such as a census block. Roads, census blocks, county and state boundaries are more common examples of spatial data and are usually depicted in a map. A Geographic Information System (GIS) is often used to store, retrieve, and render this Earth-relative spatial data. When rendered as a map, this spatial data depicts the locations of the objects on a two-dimensional piece of paper or a video monitor.

Other types of spatial 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 printed circuit boards. The differences among these 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 road builder's blueprints.

These applications all store, retrieve, update, or query some collection of features that have both non-spatial and spatial attributes. Examples of non-spatial attributes are customer name, address, orders, and part_number. The spatial attribute, such as a geocoded address or sales_region, is a coordinate geometry (or vector-based) representation of the shape of the feature. Also, the geometry is an ordered sequence of one or more vertices. The structure and semantics of the geometry are determined by its type which may be one of point, line, or polygon and are described in detail in following sections.

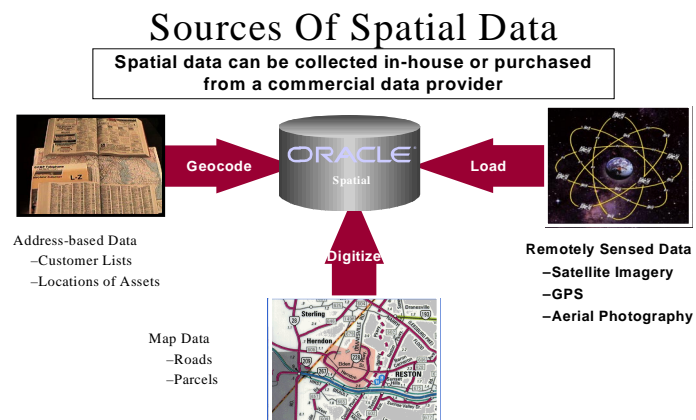


Figure 1 Spatial Data Comes from Many Sources

2.0 SPATIAL DATA AND ORDBMS

2.1 Object-Relational Database Management Systems (ORDBMS)

Traditionally, database management systems are broadly divided into two categories: relational and object-oriented. Recently, a third type of database system has become prominent. These systems combine the best of both the relational and object oriented databases. These are called the Object Relational database systems (ORDBMS). Object-relational database systems facilitate the definition, storage, retrieval, and manipulation of user-defined data types in the database through the use of user-defined functions and index methods. Thus an ORDBMS can now handle spatial information represented using a spatial object data type, and accessed or manipulated using spatial index methods and functions. Because spatial is now just another attribute represented in the database, users can use it as another qualifier or criteria when searching or browsing the database.

There are several benefits to managing the spatial and attribute data in a single database. Key benefits of this approach to spatial data management include:

- Better data management for spatial data. Users gain access to full function spatial information systems based on industry standards with an open interface to their data (e.g., SQL).
- Spatial data is now stored in enterprise-wide database, thereby facilitating spatially enabling many more applications.
- Reduced complexity of systems management by eliminating the hybrid or file based architectures of traditional GIS-based data management schemes.
- Proprietary data structures are avoided by using an open SQL platform thereby allowing for the seamless integration of e-business and location-based services. This facilitates the task of delivering applications that meet the increasingly demanding analysis and reporting needs of a growing information and knowledge management community.

2.2 Challenges of Spatial Databases

Unlike traditional database applications, spatial applications require that databases understand more complex data types like points, lines, and polygons. Also the operations on these types are complex when compared to the operations on simple types. Hence we need new technology to handle spatial data. There are four main properties of the spatial data which set it apart from traditional relational data.

2.2.1 Geometry

Geometry is a main property in any kind of spatial data. Geometry deals with the mathematical properties of an object. These properties include measurement

(metric), relationships of points, lines, angles, surfaces, and solids (topology), and order. A simple geometry is usually constructed from geometric primitives such as points, lines, curves, and areas. Complex geometries are constructed from collections of simple geometries. In addition, there are a number of geometric relationships between geometries that are important in handling spatial data. For example on a road map, a connectivity relation describes how one intersection is connected to another intersection (i.e., how two geometries interact). Metric relationships deal with the distances between two geometries. For example, find all cities located within 10 miles of a given road.

2.2.2 Distribution of Objects in Space

Usually spatial objects are very irregularly distributed in space. Consider the case where we model the town halls of all the cities in the United States as spatial objects (points). The distribution of cities on the east coast is very dense compared to the distribution of cities in Arizona and Nevada, which is sparse. In addition, different objects have largely varying extents. For example, if we look at the road network model which models roads with lines and cities with polygons, we see that there are some very large objects in the model (large roads like I-95) and small objects (small cities like Nashua, NH).

2.2.3 Temporal Changes

Spatial data often has an associated temporal property. An example is a navigation system which helps travelers find directions from place A to B in a major city. If there is an accident and some road is temporarily closed, the system has to incorporate this new data and recompute a suitable path from point A to B. When the road is opened for traffic, this new information has to be taken into account in the path computations.

2.2.4 Data Volume

Several GIS applications deal with very large databases of the order of terabytes. For example, remote sensing applications gather terabytes of data from satellites every day. Similarly, data warehousing applications and NASA's Earth Observation System are other examples of systems with terabytes of spatial data.

2.3 Requirements of a Spatial Database System

Any database system that attempts to deal with enterprise GIS and location-based services has to provide the following features:

1. A set of spatial data types to represent the primitive spatial data types (point, line, area), complex spatial data types and operations on these data types like intersection and distance.
2. The spatial types and operations on them should be part of the standard query language that is used to access and manipulate non spatial data in

the system. For example, in case of relational database systems SQL should be extended to support spatial types and operations.

3. The systems should also provide performance enhancements such as indexes to process spatial queries (range and join queries), parallel loads and queries, which are available for nonspatial data.

Any spatial database system further addresses the following five main areas to support spatial applications:

- Classification of Space
- Data Model
- Query Language
- Query Processing
- Data Organization and Indexing.

2.3.1 Classification of Space

Space is a framework to formalize specific relationships among a set of spatial objects. Depending on the relationships of interest, different models of space such as topological space, network space and metric space can be used. Topological space uses the basic notion of a neighborhood and points to formalize relationships which are invariant under elastic deformation. Topological relationships include closed, within, connected, and overlaps. Network space deals with such relationships as shortest paths, and connectivity. Metric spaces formalize the distance relationships using positive symmetric functions that obey the triangle inequality. The taxicab or “Manhattan distance is one such metric relationship. Given three points A, B, and C, the Manhattan distance from A to C is less than or equal to the sum of the distances from A to B and B to C.

2.3.2 Data Model

Object relational databases provide a higher abstraction of spatial data by incorporating concepts closer to humans’ perception of space. This is accomplished by incorporating the object oriented concept of a user-defined abstract data type (ADT) and associated functions. For example if we have land parcels stored in a database then an ADT would be a combination of the data type polygon and some associated function, such as adjacent(), which may be applied to land parcels to determine if they are touching.

2.3.3 Query Language

It is clear that traditional SQL is inadequate to express typical spatial queries. This has prompted various efforts to extend its capability with spatial-friendly constructs. At the same time, various standards committees (most notably OGC)

are working on specifications that extend SQL with the generic functionality offered by object-relational database management systems.

2.3.4 Spatial Query Processing

Spatial queries are often processed using filter and refine techniques. In the first filter step, an approximate representation of a spatial object is used to determine a set of candidate objects that are likely to satisfy the given spatial query. The approximations are chosen such that if the approximations of objects A and B do satisfy a relationship, then the objects A and B are likely to have that spatial relationship. For example, if the approximations are disjoint then the objects A and B will be disjoint. If the approximations are non-disjoint, however, objects A and B may still be disjoint.

There are several advantages to applying this filter and refine strategy. First, spatial objects tend to be very large and hence consume considerable amounts of main memory. An approximate representation of a spatial object takes considerably less space and time to load into memory. Second, computations on spatial objects tend to be very complex and computationally expensive. The more complex the objects, the more processing is required to compute spatial relationships. Computations using approximate objects tend to be very fast and require far less computational cycles.

Spatial queries are classified into two categories: (i) Window queries and (ii) Join queries. Window queries take a spatial object (called the window object) and seek spatial objects from a table that satisfy a binary relation with the window object. For example, the query, “Find all the roads in the City of Minneapolis which overlap the park Minnetonka” is a window query. Here the object representing the park Minnetonka is the window object. A join query seeks all the object pairs from two tables which satisfy a given relation. For example, the query, “Find all the roads and parks which overlap from the city of Minneapolis” is a join query.

2.3.5 Spatial Indexing

Indexes help speed up the execution of SQL statements in the database by providing a faster access path to data. Spatial indexes are also the primary means of reducing disk I/O when manipulating the data. Databases provide standard indexing mechanisms that work with scalar data, these indexes are not suitable for spatial data.

The main purpose of a spatial index is to facilitate spatial selection. That is, in response to a query, the spatial index will only search through a subset of objects embedded in the space to retrieve the answer set. There are essentially two ways of providing a spatial index: (1) Dedicated external spatial data structures are added to the system, offering for spatial data what a B-tree does for standard attributes, and (2) spatial objects are mapped into a 1-D space so that they can be stored within a standard 1-D index such as the B-tree. Apart from spatial

selection, spatial indexing also supports other operations such as spatial joins, finding the object closest to a query value, and so forth.

3.0 ORACLE SPATIAL

Oracle Spatial provides a completely open architecture for the management of spatial data within a database management system. The functionality provided by Oracle Spatial is completely integrated within the database server. Users define and manipulate spatial data through SQL and gain access to standard Oracle features such as a flexible n-tier architecture, object capabilities, robust data management utilities, Java stored procedures. This ensures data integrity, recovery, and security features that are virtually impossible to obtain with hybrid architectures.

3.1 Spatial Data Modeling

Oracle Spatial supports three geometric primitive types and geometries composed of collections of these types. The three primitive types are: Point, Line String, and N-point polygon where all these primitive types are in 2-Dimensions. A 2-D point is an element composed of two ordinates, X and Y. Line strings are composed of an ordered sequence of two or more points that define line segments. Line strings can be composed of straight line segments, arc segments or a mixture of both. Polygons are composed of connected line strings that form a closed ring and the interior of the polygon is implied. Because polygons are composed of line strings, this implies that a polygons can have some edges as straight lines and some edges as circular arcs.

The spatial data model is a hierarchical structure consisting of elements, geometries, and layers. Spatial layers are composed of geometries that are in turn composed of elements.

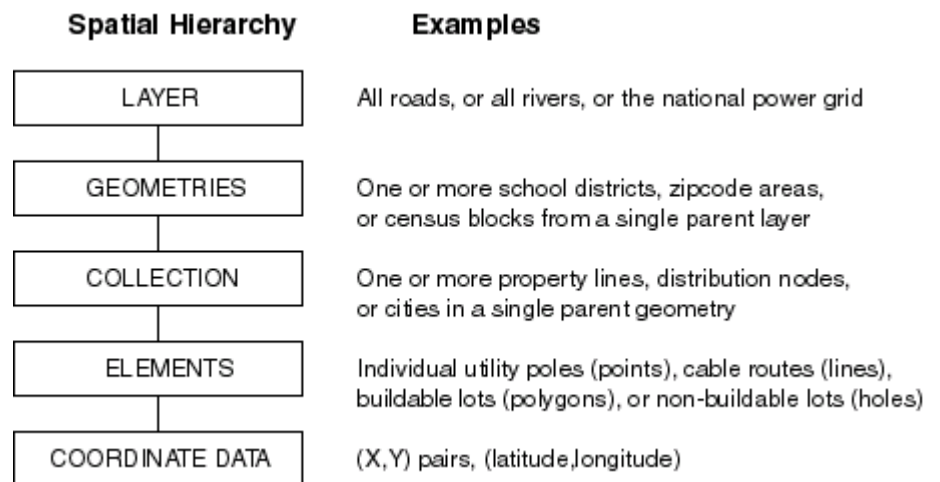


FIGURE 2. Data Model Hierarchy

An element is the basic building block of a geometry. For example, elements might model utility poles (points), roads (line strings), or county boundaries (polygons). In the case of polygons with holes (such as an island within a lake) the exterior ring and the interior ring of the polygon are considered as two distinct elements that together make up a complex polygon. A geometry is the representation of a user's spatial feature, modeled as an ordered set of primitive elements.

A geometry can consist of a single element or a homogeneous or heterogeneous collection of primitive types. A multipolygon, such as one used to represent a set of islands, is a homogeneous collection. A heterogeneous collection is one in which the elements are of different types.

A layer is a heterogeneous collection of geometries which share the same attribute set. For example, one layer in a spatial information system might include topographical features, while another describes population density, and a third describes the network of roads and bridges in an area. Layers correspond to a table or set of tables while a geometry is one instance of the type `MDSYS.SDO_GEOMETRY` and is stored in a particular row and column in a table.

The type `MDSYS.SDO_GEOMETRY` is a container for storing points, lines, polygons, or homogeneous or heterogeneous collections of these elements. Attributes consist of a geometry type identifier, a spatial reference system identifier, an element descriptor array, and an ordinate array among other things. The ordinate array contains the values for coordinate pairs or triples that define the vertices of the geometric elements. The element descriptor array defines how these ordinates should be assigned to the element or elements that constitute the geometry. This array also determines whether a pair (or triple) of vertices is connected by a straight line segment or a circular arc. `Arclinestring` and `Arcpolygon` are elements whose vertices are connected using circular arcs. A compound element is one whose vertices are connected by a mix of straight line and circular arc segments.

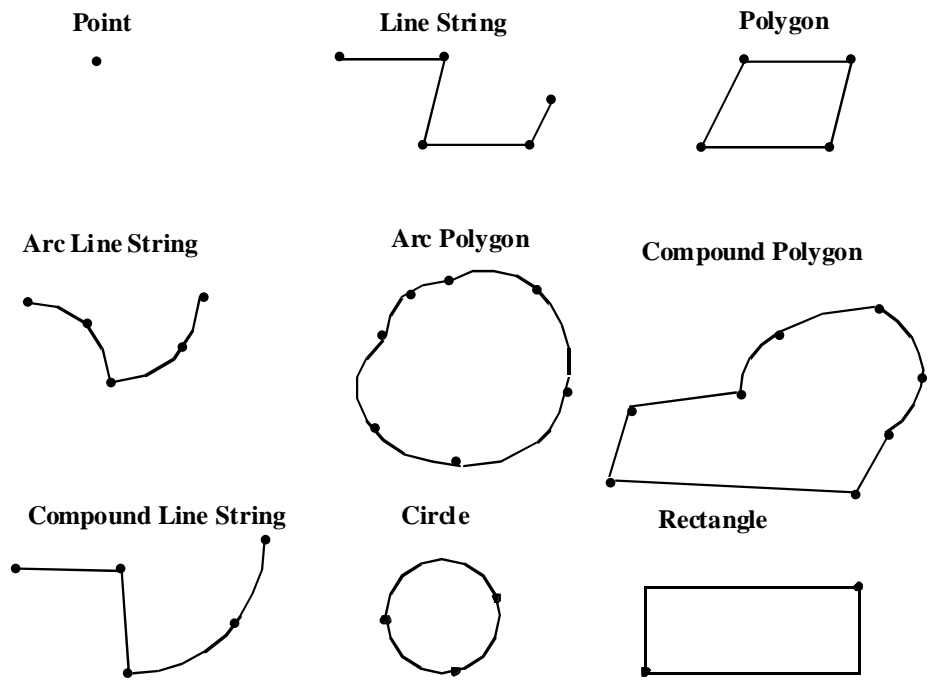


FIGURE 3. Examples of shapes representable using SDO_GEOMETRY.

A column in a table can be declared as being of type MDSYS.SDO_GEOMETRY. For example, you can create a table named ROADS as follows:

```
CREATE TABLE ROADS (
  Name          Varchar2(64),
  Classification Varchar2(64),
  Geometry      MDSYS.SDO_GEOMETRY)
```

Rows are inserted using the standard SQL INSERT statement as follows (Line example):

```
INSERT INTO ROADS VALUES('Short Street', 'Bylane',
  MDSYS.SDO_GEOMETRY(2002, 8307, null,
  MDSYS.SDO_ELEM_INFO_ARRAY(1, 2, 1),
  MDSYS.SDO_ORDINATE_ARRAY(10, 10, 10, 15, 15, 15)));
```

The first value, 2002, is a geometry type identifier that indicates this is a 2D linear geometry. The second value is 8307 specifying that this geometry has the spatial reference system identified by 8307 (in this case 8307 corresponds to WGS 84). The third value is set to null, but it can be used as a label point location. The fourth value is the element descriptor array and specifies that the

element's ordinates start at offset 1 in the SDO_ORDINATES array, that the element is a line string (indicated by typecode 2), and that its vertices are connected by straight line segments. The fifth value is the ordered sequence of ordinate values which in this instance specifies the vertices {(10, 10), (10, 15), (15, 15)}.

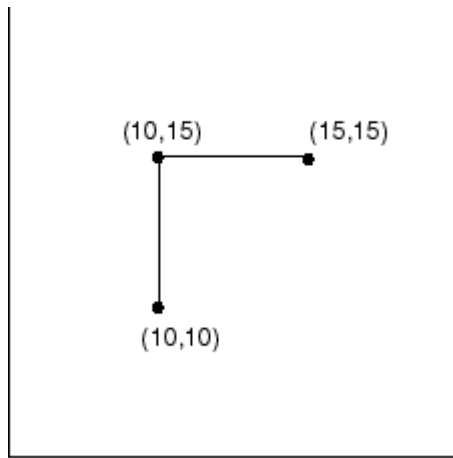


FIGURE 4. The line representing Short Street.

In Oracle Spatial, a layer is defined as a column, table pair in the database. And all geometries in a layer must share some of the spatial attributes. Specifically, the dimensions and the spatial reference system for all geometries in a layer should be the same. Even though this information is available with each geometry, some times it is more efficient to retrieve this information from a database view instead of retrieving it from each geometry. Hence this information is also stored as part of a view defined as:

```
VIEW USER_SDO_GEOM_METADATA (
  TABLE_NAME          VARCHAR2 ( 32 ) ,
  COLUMN_NAME          VARCHAR2 ( 32 ) ,
  DIMINFO              MDSYS.SDO_DIM_ARRAY ) ,
  SRID                 NUMBER ) ;
```

where SDO_DIM_ARRAY is a VARRAY(4) of MDSYS.SDO_DIM_ELEMENT, and SDO_DIM_ELEMENT is defined as

```
CREATE TYPE MDSYS.SDO_DIM_ELEMENT AS OBJECT (
  SDO_DIMNAME          VARCHAR2 ( 32 ) ,
  SDO_LB               NUMBER, -- Lower bound
  SDO_UB               NUMBER, -- Upper bound
  SDO_TOLERANCE        NUMBER ) ;
```

For each table.column that is of type MDSYS.SDO_GEOMETRY there should be one entry in the USER_SDO_GEOM_METADATA table that contains the dimension information for that column, along with the SRID if one is required. For the ROADS.GEOMETRY column above, for example, the following INSERT operation should be done before creating any spatial index or using any spatial functions or operators.

```
INSERT INTO SDO_GEOM_METADATA VALUES ( 'ROADS' ,  
    'GEOMETRY' , MDSYS.SDO_DIM_ARRAY(  
        MDSYS.SDO_DIM_ELEMENT( 'X' , 0 , 100 , 0.05 ) ,  
        MDSYS.SDO_DIM_ELEMENT( 'Y' , 0 , 100 , 0.05 ) ) ;
```

The dimensionality of the geometries stored in a particular Table.Column can then be determined by looking at the corresponding diminfo in USER_SDO_GEOM_METADATA. One possible SQL statement for this is:

```
SELECT COUNT( * ) FROM  
    THE( SELECT DIMINFO FROM SDO_GEOM_METADATA  
        WHERE TABLE_NAME = 'ROADS' ) ;
```

Further details on the attributes of MDSYS.SDO_GEOMETRY and its use in representing points, lines, polygons, circles, rectangles, circular arcs and other vector geometric shapes can be found in the *Oracle Spatial User's Guide and Reference*.

3.2 Operations on the Spatial types

A spatial relation between entities is one based upon the location of their associated geometries. The most common spatial relations are based on topological and distance criteria.

The binary topological relationships between two spatial objects A and B in the Euclidean space is based how the two objects A and B interact with respect to their interior, boundary and exterior. This is called the 9-intersection model for the topological relationships between two objects. This can be concisely represented using a 3-by-3 matrix. From this matrix, one can theoretically distinguish between $2^9=512$ binary relationships between A and B. In case of polygonal 2-dimensional objects, only eight relations can be realized which provide mutually exclusive and complete coverage for A and B. These relationships are *contains*, *coveredby*, *covers*, *disjoint*, *equal*, *inside*, *overlap*, and *touch*.

Oracle Spatial supports this 9-intersection model for determining the topological relationships between two objects. Other relationships can be derived as a

combination of the above eight relations. For example, OVERLAPBDYDISJOINT can be defined as the relation where the objects overlap but the boundaries are disjoint. This functionality is made available through an operator, SDO_RELATE, and a function, SDO_GEOM.RELATE(). The operator, SDO_RELATE, is registered with the extensible optimizer and hence the optimizer will evaluate various query plans that include or exclude the use of a spatial index. The function, SDO_GEOM.RELATE, does not use the spatial index and simply evaluates the two geometries that are passed to it via the argument list for the specified topological relationship. For example, to find all the parks in a city which overlap the rivers in the city, one can formulate two slightly different SQL queries:

```
SELECT parks.name FROM parks, rivers WHERE
  mdsys.sdo_relate(parks.geometry, rivers.geometry,
    'mask = OVERLAPBDYINTERSECT') = 'TRUE';

SELECT parks.name FROM parks, rivers WHERE
  sdo_geom.relate(parks.geometry, rivers.geometry,
    'OVERLAPBDYINTERSECT') = 'OVERLAPBDYINTERSECT';
```

The first uses the operator SDO_RELATE and will likely be evaluated using relevant spatial indices. The second uses the function SDO_GEOM.RELATE and hence the query will be evaluated on a row by row basis using full table scans.

Other index-aware operators defined by Oracle Spatial are:

- SDO_FILTER(Geometry1, Geometry2, otherParams): Uses only the index entries to determine which geometry pairs are likely to be non-disjoint.
- SDO_WITHIN_DISTANCE(Geometry1, Geometry2, 'distance = NN'): Determines if geometry2 is within NN (using Euclidean distance computation) units from Geometry1. It does so by first computing the buffer polygon of NN units around geometry 2 and then comparing index entries for geometries 1 and the buffer polygon. If the index entries match then the intersection between geometry 1 and the buffer polygon is evaluated to determine the final result.
- SDO_NN(Geometry1, Geometry2, 'num_res=NN'): Determines the NN nearest neighbors from Geometry2 to all the geometries corresponding to the table from Geometry1.

There is also a function SDO_GEOM.SDO_BUFFER which returns the buffer polygon of a given distance around the specified geometry. Also provided are set operations like UNION, INTERSECTION, DIFFERENCE and SYMMETRIC-DIFFERENCE. For example, given two polygonal spatial

objects A and B, one can compute and return a new object C which is the UNION of A and B.

For details of the syntax and usage of the above operators and functions, see the *Oracle Spatial User's Guide and Reference*.

3.3 Spatial Indexing

The introduction of spatial indexing capabilities into the database engine through Oracle Spatial is a key feature. A spatial index is a mechanism to limit searches within tables (or data spaces) based on spatial criteria. An index is required to efficiently process queries such as finding objects within a data space that overlap a query area. This is defined by a query polygon (fence locate). A second type of query (spatial join) is finding pairs of objects from within two data spaces that spatially interact with one another. Oracle Spatial provides a linear quadtree based indexing scheme and an R-tree based indexing scheme for indexing spatial data.

Quad-Tree Indexing: A linear quad-tree index maps geometric objects to a set of numbered tiles. A tile in 2-D space is a box whose edges are orthogonal to the two coordinate axes. The coordinate space where all the geometries are located is decomposed in a regular hierarchical manner. The range of coordinates (the coordinate space) is viewed as a rectangle. At the first level of the decomposition, this rectangle is divided in half in each of the coordinate directions, forming four sub-tiles called quads. At each subsequent level, each quad is divided in half in each coordinate direction forming 4 subtiles. As the name implies, the 4-leaf structure of a quadtree can be used to build an index tree. This process continues until some termination criteria, such as the size of the tiles is met. These tiles can be linearly ordered using the z-ordering or equivalent scheme resulting in a linear quadtree.

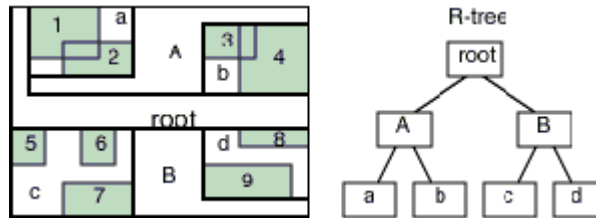
R-Tree Indexing: Oracle Spatial includes R-tree indexing, in addition to quadtree indexing capability. R-tree indexes can be used in place of quadtree indexes, or in conjunction with them. In addition, R-tree indexing can be used for any 3D and 4D indexing of data – critical to solving problems in oil exploration, architecture, engineering, and many other scientific applications.

An R-tree index approximates each geometry with the smallest single rectangle that encloses the geometry (called the minimum bounding rectangle, or MBR).



For a layer of geometries, an R-tree index consists of a hierarchical index on the minimum bounding rectangles of the geometries in the layer. Because R-tree indexes are fast and work directly on geodetic data they are the preferred indexing mechanism for working with spatial data. Geodetic data is data

consisting of coordinates that are defined relative to a particular representation of the figure of Earth, or datum.



Oracle Spatial uses the Extensibility Framework of Oracle9i to tightly integrate the interfaces for spatial index creation and query processing with the SQL Engine. The syntax for index creation and maintenance are simply domain specific extensions to the CREATE, ALTER, and DROP statements. Creating a spatial index on a table (ROADS) with a column (GEOMETRY) of type MDSYS.SDO_GEOMETRY can be done as follows:

```
CREATE INDEX ROADS_GEOMIDX ON ROADS(GEOMETRY)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

The index can be altered:

```
ALTER INDEX ROADS_GEOMIDX REBUILD
  PARAMETERS ('tablespace=DOM_IDX');
```

or deleted:

```
DROP INDEX ROADS_GEOMIDX;
```

For further details on the syntax and usage of the above SQL statements, see the *Oracle Spatial User's Guide and Reference*.

3.4 Query Processing

Queries and data manipulation statements can involve application-specific operators, like the Relate operator in the spatial domain. In general, user-defined operators are bound to functions. This is called the functional implementation of an operator. For example, consider the following query

```
SELECT a.feature FROM ROADS a, windows b WHERE
  MDSYS.SDO_RELATE(a.geometry, b.geometry,
    'mask=OVERLAPBDYINTERSECT');
```

This example finds all the features in the ROADS table that overlap with the window query objects in the windows table. In the case of the functional implementation of the operator, each geometric object in the ROADS table is

tested against each window object to find out whether or not they overlap. This could be very costly if the number of objects in the ROADS table is high. However, operators can also be evaluated using indexes, for instance, the equality operator for scalar data types can be evaluated using a B-tree index. The query optimizer understands the predefined equality operator, and the B-tree index method, and therefore knows that equality can be evaluated using a B-tree index. In traditional relational database systems, it was not possible to create domain specific indexes that could be used in evaluating application specific operators. Moreover, the optimizer is not extensible, which would let it know the cost of evaluating an application specific operator. To facilitate the evaluation of application specific operators, Oracle9i provides three different extensions to the server:

- SQL is extended so that user-defined operators can be expressed in a similar fashion as predefined operators.
- Index creation, scan, and maintenance are extended so that applications can define and manage their specialized indexes.
- The optimizer is extended so that applications can inform the optimizer about the evaluation cost and selectivity of its specialized operators. This functionality is termed Extensible Indexing in Oracle9i.

Extensible indexing: With extensible indexing, the application defines the structure of the domain (specific) index. The application stores the index data either inside the Oracle database (in tables) or outside the Oracle database (in the form of files). It manages, retrieves and uses the index data to evaluate user queries. In effect, the application controls the structure and semantic content of the domain index. The database system interacts with the application to build, maintain, and employ the domain index.

It is highly desirable that domain index data is stored in the database, thereby allowing it to handle the physical storage. Then index data will be treated like scalar data in the database and inherits all the benefits of the database server. The main advantage of this extensible indexing framework is that the index is always in sync with the data. Once the index is built, all inserts, updates, or deletes on the base table will automatically result in updates of the index data. Once the domain index is built, it is treated like a regular index. The server knows the existence of the index and thus manages all the index related work using user-defined functions. This is accomplished by registering the domain index method and operators with the extensible optimizer.

Extensible Optimizer: The optimizer generates an execution plan for any SQL statement. For simplicity, we will describe the control flow in an optimizer for a SELECT statement, but the same methods apply for other data manipulation statements. An execution plan includes an access method for each table, and an ordering (called the join order) of the tables, in the FROM clause. System-defined access methods include indexes, hash clusters, and table scans. The

optimizer chooses a plan by generating a set of join orders or permutations, computing the cost of each, and selecting the one with the lowest cost.

For each table in the join order, the optimizer computes the cost of each possible access method and join method and chooses the one with the lowest cost. The cost of the join order is the sum of the access method and join method costs. The costs are calculated using algorithms which together compose the cost model. A cost model can include varying level of detail about the physical environment in which the query is executed.

The optimizer uses statistics about the objects referenced in the query to calculate costs and selectivities. The selectivity of a predicate is the fraction of rows in a table that will be chosen by the predicate. It is a number between 0 and 1. The selectivity of a predicate is used to estimate the cost of a particular access method. The optimizer estimates the cost of various access paths to choose an optimal plan. For example, it computes the cost of using an index and a full table scan to choose between the two. For domain indexes, however, because the optimizer does not know the internal storage structure of the index, it cannot compute a good estimate of the cost of a domain index. It therefore defers to the application to obtain these estimates.

Indextypes can be used to bind user-defined operators to an index. This results in any SQL query containing those operators being efficiently evaluated using the corresponding index. Oracle Spatial defines an INDEXTYPE (SPATIAL_INDEX) and associated OPERATORS (SDO_RELATE, SDO_FILTER, SDO_WITHIN_DISTANCE, and SDO_NN). Consider a query such as the following:

```
SELECT Parks.Name FROM Parks, Roads WHERE
  MDSYS.SDO_RELATE(Parks.Geometry, Roads.Geometry,
    'MASK=ANYINTERACT') = 'TRUE'
AND Roads.Name = 'I-93';
```

The optimizer can determine that it is less expensive to first evaluate the second predicate (Roads.Name = 'I-93') followed by SDO_RELATE(...) = 'TRUE' rather than the other way around.

For further details on the syntax and usage of spatial operators and notes on spatial query processing, see the *Oracle Spatial User's Guide and Reference*.

4.0 DIFFERENCES BETWEEN SPATIAL RELEASE 8i AND 9i

Oracle9i supports new features which extend the range and productivity of application developers, enabling a broader range of applications and improving performance.

4.1 Spatial Aggregates

SQL has long had aggregate functions, which are used to summarize the results of an SQL query. The following example uses the SUM aggregate function to aggregate employee salaries by department:

```
SELECT SUM(salary), dept
       FROM employees
       GROUP BY dept;
```

Oracle9i adds aggregate functions which operate on the results of SQL queries involving geometry objects. Spatial aggregate functions return a geometry object of type SDO_GEOMETRY. For example, the following statement returns the minimum bounding rectangle of all the geometries in a table:

```
SELECT SDO_AGGR_MBR(c.shape)
       FROM cola_markets c;
```

The use of spatial aggregates improves performance and reduces the complexity of the underlying code base. Before 9I, a block of PL/SQL code would be required to get the result of the above SQL statement and hence it is not easy to use. In 9I, this complexity is reduced making it easy to get Spatial aggregate results. Other aggregate functions defined in Spatial are: SDO_AGGR_UNION, SDO_AGGR_CENTROID, SDO_AGGR_CONVEXHULL, etc. For a complete listing, see the *Oracle Spatial User's Guide and Reference*.

4.2 Function-Based Index Support

A function-based spatial index facilitates location-based queries on relational attributes without explicitly storing location information as a column of type SDO_GEOMETRY. Users can create spatial indexes on spatial data stored in relational columns (for example in columns of longitude and latitude). This spatial index will make it possible to invoke spatial operators on these relational columns without the need to create an SDO_GEOMETRY column.

This is useful for business intelligence applications which have a schema for storing location data but are not inclined to change their current schema to move the spatial data to a column of type SDO_GEOMETRY. In addition, spatial indexing is also supported for objects that embed SDO_GEOMETRY type as an attribute. For example, one can define a new address_type object with address as one attribute and location as another attribute with location as an SDO_GEOMETRY object. Then the address_type object can be spatial indexed and can be used in all spatial functions and operators.

4.3 Geodetic Coordinate Support

With Oracle9i, Spatial provides a rational and complete treatment of geodetic coordinates. Before Oracle 9i, Spatial computations were based solely on flat (Cartesian) coordinates, regardless of the coordinate system specified for the layer of geometries. Consequently, computations for data in geodetic coordinate systems were inaccurate, because they always treated the coordinates as if they were on a flat surface, and they did not consider the curvature of the earth's surface. With the current release, ellipsoidal surface computations consider the curvatures of arcs in the specified geodetic coordinate system and return correct, accurate results. In other words, with the current release, Spatial queries return the right answers all the time. In addition, all Spatial functions fully support distance, area, and angular units.

4.4 Partitioning Support for Spatial Indexes

Another new feature of Oracle9i is the ability to partition spatial indexes in association with partitioned tables. Partitioned spatial indexes can provide the following benefits:

- reduced response times for long-running queries; partitioning reduces disk I/O operations
- reduced response times for concurrent queries; I/O operations run concurrently on each partition
- easier index maintenance, because of partition-level create and rebuild operations
- indexes on partitions can be rebuilt without affecting the queries on other partitions
- storage parameters for each local index can be changed independent of other partitions.

4.5 Linear Referencing Support

Linear referencing is a natural and convenient means to associate attributes or events to locations or portions of a linear feature. It has been widely used in transportation applications (such as highways, railroads, and transit routes) and utilities applications (such as gas and oil pipelines). The major advantage of linear referencing is its capability of locating attributes and events along a linear feature with only one parameter (usually known as measure) instead of two (such as longitude/latitude or x/y in Cartesian space). Sections of a linear feature can be referenced and created dynamically by indicating the start and end locations along the feature without explicitly storing them.

The linear referencing system (LRS) application programming interface (API) in Oracle Spatial provides server-side LRS capabilities at the cartographic level. The linear measure information is directly integrated into the Oracle Spatial

geometry structure. The Oracle Spatial LRS API provides support for dynamic segmentation, and it serves as a groundwork for third-party or middle-tier application development virtually for any linear referencing methods and models in any coordinate systems.

4.6 Performance Enhancements

Oracle9i provides significant improvements to spatial data storage and indexing performance over that of previous releases. R-tree indexes can now be created up to 20% faster than with the previous release. Partitioning spatial data and using partitioned local indexes can provide additional performance gains for queries on large data sets, and concurrent queries and updates. Spatial aggregate functions speed retrieval of large sets of SDO_GEOMETRY objects.

Spatial queries that utilize secondary filters will run significantly faster when using all masks other than ANYINTERACT. Gains of up to 200% can result, depending on the complexity of the geometries. WITHIN_DISTANCE queries can run up to 40% faster when using R-tree indexes, and the VALIDATE_GEOMETRY function runs up to 200% faster depending on the complexity of the geometries.

For more information on performance characteristics, see a separate performance technical white paper on the Oracle Technology Network at <http://otn.oracle.com/products/spatial>.

5.0 SUMMARY

Oracle Spatial Release 9i provides a completely open architecture for the management of spatial data within a database management system. The functionality provided by Oracle Spatial is completely integrated within the database server. Users define and manipulate spatial data through SQL and gain access to standard Oracle features, such as a flexible n-tier architecture, object capabilities, Java stored procedures, and robust data management utilities, ensuring data integrity, recovery, and security features.



Oracle Spatial

May 2001

Author: Dr. Jayant Sharma

Contributing Authors: Dr. John Herring, Siva Ravada

Oracle Corporation

World Headquarters

500 Oracle Parkway

Redwood Shores, CA 94065

U.S.A.

Worldwide Inquiries:

Phone: +1.650.506.7000

Fax: +1.650.506.7200

www.oracle.com

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.

Copyright © 2001 Oracle Corporation

All rights reserved.