CHAPTER 2

Overview of Oracle Spatial

To run the examples in this chapter, you need to load three datasets in the `spatial` schema as follows. Please refer to the Introduction for instructions on creating the `spatial` schema and other setup details.

```
imp spatial/spatial file=gc.dmp ignore=y full=y
imp spatial/spatial file=map_large.dmp tables=us_interstates
imp spatial/spatial file=map_detailed.dmp tables=us_restaurants
```

In Chapter 1, we observed that spatial information can add value to a range of different applications. We examined the benefits of storing spatial information together with other data in the database.

The Spatial technology suite in Oracle enables storage of spatial data in the database and facilitates different types of analyses on spatial data. This chapter provides an overview of the Spatial technology suite and covers the following topics:

- An overview of the Oracle Spatial architecture and technology.
- An examination of the *functionality* of different components of this Spatial technology suite in Oracle. This includes a brief introduction to the data type that stores spatial data (`SDO_GEOMETRY`), the query predicates for performing spatial query and analysis, and additional functionality to perform visualization.
- A description of how this functionality is *packaged* into different products that are shipped with different editions of Oracle software. We will discuss the relative merits of each product in turn.
- What to expect in a typical install of Oracle Spatial. This knowledge should get you off to a smooth start in spatially enabling your application.

Technology and Architecture Overview

Oracle Spatial technology is spread across two tiers: the Database Server and the Application Server. Figure 2-1 depicts the various components that comprise Oracle's spatial technology stack and indicates the distribution of the components across the Database Server and Application Server tiers. Basic components that are provided as part of Oracle Database Server 10*g* include storage model, query and analysis tools, and location-enabling/loading utilities. The MapViewer component is provided in Oracle Application Server 10*g*.

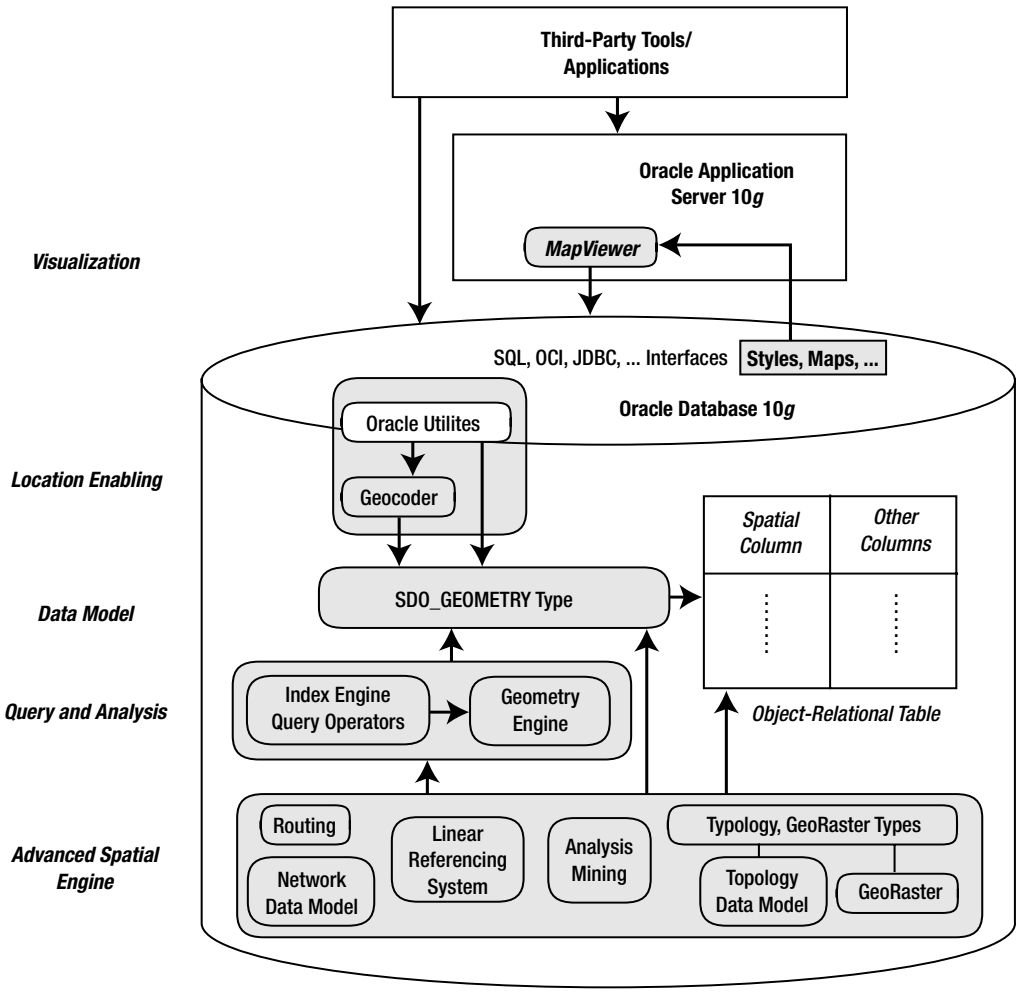

Figure 2-1. *Oracle Spatial technology components*

The basic components from Figure 2-1 can be described as follows:

- *Data model*: Oracle Spatial uses a SQL data type, SDO_GEOMETRY, to store spatial data inside an Oracle database. Users may define tables containing columns of type SDO_GEOMETRY to store the locations of customers, stores, restaurants, and so on, or the locations and spatial extents of geographic entities such as roads, interstates, parks, and land parcels. This data type is described in detail in Chapter 4.
- *Location enabling*: Users can add SDO_GEOMETRY columns to application tables. This process is described in detail in Chapter 3. Users can populate the tables with SDO_GEOMETRY data using standard Oracle utilities such as SQL*Loader, Import, and Export. This process is described in Chapter 5. Alternatively, users can convert implicit spatial information, such as street addresses, into SDO_GEOMETRY columns using the geocoder component of Oracle Spatial, as described in Chapter 6.
- *Spatial query and analysis*: Users can query and manipulate the SDO_GEOMETRY data using the query and analysis component, comprising the Index and Geometry Engines. Full details of this process are given in Chapters 8 and 9.
- *Advanced Spatial Engine*: This component comprises several components that cater to sophisticated spatial applications, such as GIS and bioinformatics. This includes, for example, the GeoRaster component that allows storage of spatial objects using images (groups of pixels) rather than points, lines, and vertices. These components are covered in Appendixes A through D.
- *Visualization*: The Application Server components of Oracle's spatial technology include the means to visualize spatial data via the MapViewer tool. MapViewer renders the spatial data that is stored in SDO_GEOMETRY columns of Oracle tables as displayable maps. This feature is described in detail in Chapter 11.

Note also in Figure 2-1 that third-party tools may access spatial data either through the Application Server or directly from the database using SQL, OCI, JDBC, or other appropriate interfaces. Programming with spatial data via these APIs is described in Chapter 7.

■ **Note** The core subset of this functionality (known as the *Locator* component) is included for free in all editions of the database (essentially, the SDO_GEOMETRY data type and the Index Engine). The rest of the components, along with the data type and the Index Engine, are packaged in a priced option of the Enterprise Edition of the database (known as the *Spatial* option). We discuss this in more detail later in this chapter.

In the following sections, we'll preview these components and introduce you to some (very basic) SQL to create a table that stores spatial data, to populate that data, and to perform simple proximity analysis. All of these topics are covered in full detail in subsequent chapters, as described previously, but this should serve as a useful introduction to the technology and help you to get started.

Getting Started with Oracle Spatial

Oracle Spatial technology is automatically installed with the Standard or Enterprise Edition of an Oracle database server. So, as long as you have one of these editions of version 10.1.0.2 or higher, you should be able to work through the simple examples in the coming sections. If you encounter any problems, you can refer to the "What to Expect in an Oracle Spatial Install" section at the end of this chapter. Note that the Database Server license includes only a few of the functions described in this section. To use the rest of the functionality, you should obtain a separate product license for the Spatial option.

Data Model: Storing Spatial Data

In Chapter 1, we briefly discussed the idea that spatial information is specified using two components: a *location* with respect to some origin and a geometric *shape*.

- *Location* specifies where the data is located with respect to a two-, three-, or four-dimensional coordinate space. For instance, the center of San Francisco is located at coordinates (–122.436, –37.719) in the two-dimensional "latitude, longitude" space.
- *Shape* specifies the geometric structure of the data. Point, line, and polygon are examples of possible shapes. For instance, the center of San Francisco is located at coordinates (–122.436, –37.719) in the two-dimensional "latitude, longitude" space and is a *point* shape. Note that point specifies both a location and a default shape. Alternately, shape could specify a *line* or a *polygon* connecting multiple points (specified by their locations). For instance, the city boundary of San Francisco could be a *polygon* connecting multiple *points*.

  In some applications, the shapes could be more complex and could have multiple polygons, and/or polygons containing holes. For instance, the state boundaries for Texas and California include multiple polygons and some with islands. In general, spatial information, occurring in GIS, CAD/CAM, or simple location-enabled applications, could be arbitrarily complex.

The `SDO_GEOMETRY` data type captures the *location* and *shape* information of data rows in a table. This data type is internally represented as an Oracle object data type. It can model different shapes such as points, lines, polygons, and appropriate combinations of each of these. In short, it can model spatial data occurring in most spatial applications and is conformant with the Open GIS Consortium (OGC) Geometry model.[1]

Chapter 4 provides details about what types of spatial data `SDO_GEOMETRY` can model and what it cannot, and it also covers the structure of `SDO_GEOMETRY` and the tools to construct, validate, and debug `SDO_GEOMETRY` objects. For now, it is sufficient to understand that we can create tables with `SDO_GEOMETRY` columns to store the locations of objects.

---

1. Open GIS Consortium, Inc., "OpenGIS Simple Features Specification for SQL, Revision 1.1," `http://www.opengis.org/docs/99-049.pdf`, May 5, 1999.

Location Enabling

We can create tables with the SDO_GEOMETRY columns to store locations. For instance, we can create the us_restaurants_new[2] table as shown in Listing 2-1.

Listing 2-1. *Creating the* us_restaurants_new *Table*

```
SQL> CREATE TABLE  us_restaurants_new
(
  id             NUMBER,
  poi_name       VARCHAR2(32),
  location       SDO_GEOMETRY    -- New column to store locations
);
```

Now that you know basically how to create tables to store locations, let's briefly examine the tools to populate such tables. Since SDO_GEOMETRY is an object type, just like any other object type, you can populate an SDO_GEOMETRY column using the corresponding object constructor. For example, you can insert a location of (–87, –78) for a Pizza Hut restaurant into the us_restaurants table as shown in Listing 2-2.

Listing 2-2. *Inserting a Value for the* SDO_GEOMETRY *Column in an Oracle Table*

```
SQL> INSERT INTO  us_restaurants_new  VALUES
(
  1,
  'PIZZA HUT',
  SDO_GEOMETRY
  (
    2001,  -- SDO_GTYPE attribute: "2" in 2001 specifies dimensionality is 2.
    NULL,    -- other fields are set to NULL.
    SDO_POINT_TYPE  -- Specifies the coordinates of the point
    (
      -87,  -- first ordinate, i.e., value in longitude dimension
      -78,  -- second ordinate, i.e., value in latitude dimension
      NULL  -- third ordinate, if any
    ),
    NULL,
    NULL
  )
);
```

2. Note that the us_restaurants table already exists. So we name this new table us_restaurants_new.

The SDO_GEOMETRY object is instantiated using the object constructor. In this constructor, the first argument, 2001, specifies that it is a two-dimensional point geometry (a line would be represented by 2002, a polygon by 2003, and a collection by 2004).

The fourth argument stores the location of this point in the SDO_POINT attribute using the SDO_POINT_TYPE constructor. Here, we store the geographic coordinates (–87, –78). In this example, the remaining arguments are set to NULL.

■ **Note** In Chapter 4, we examine the structure of the SDO_GEOMETRY type in detail and describe how to choose appropriate values for each field of the SDO_GEOMETRY type.

Note that the preceding example shows a single SQL INSERT statement. Data loading can also be performed in bulk using Oracle utilities such as SQL*Loader, Import/Export, or programmatic interfaces such as OCI, OCCI, and JDBC. These utilities and interfaces come in very handy when populating data from GIS vendors or data suppliers.

In some applications, spatial information is not explicitly available as coordinates. Instead, the address data of objects is usually the only spatial information available. You can convert such address data into an SDO_GEOMETRY object using the geocoder component (provided with the Spatial option). The geocoder takes a postal address, consults an internal country-wide database of addresses and locations, and computes the longitude, latitude coordinates for the specified address. This process is referred to as *geocoding* in spatial terminology. The address/location database is usually provided by third-party data vendors. For the United States, Canada, and Europe, NAVTEQ and Tele Atlas provide such data.

Listing 2-3 shows how to use the geocoder to obtain the coordinates in the form of an SDO_GEOMETRY object for the address '3746 CONNECTICUT AVE NW' in Washington, D.C.

Listing 2-3. *Converting Address Data (Implicit Spatial Information) to the* SDO_GEOMETRY *(Explicit Spatial Information) Object*

```
SQL> SELECT
SDO_GCDR.GEOCODE_AS_GEOMETRY
(
   'SPATIAL',                  -- Spatial schema storing the geocoder data
   SDO_KEYWORDARRAY            -- Object combining different address components
   (
     '3746  CONNECTICUT AVE NW',
     'WASHINGTON, DC 20008'
   ),
   'US'                        -- Name of the country
) geom
FROM DUAL ;
```

```
GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------
SDO_GEOMETRY
(
   2001,
   8307,
   SDO_POINT_TYPE(-77.060283, 38.9387083, NULL),
   NULL,
   NULL
)
```

This geocoding function, `geocode_as_geometry`, takes three arguments. The first argument is the schema. In this example, we use the `'spatial'` schema. The second argument specifies an `SDO_KEYWORDARRAY` object, composed from different components of an address. In this example, `SDO_KEYWORDARRAY` is constructed out of the street component `'3746 Connecticut Ave NW'` and the city/zip code component `'Washington, DC 20008'`. The third argument to the geocoding function specifies the `'US'` dataset that is being used to geocode the specified street address. The function returns an `SDO_GEOMETRY` type with the longitude set to –77.060283 and the latitude set to 38.9387083.

The geocoder can also perform fuzzy matching (via tolerance parameters, which we'll cover in the next chapter). In the same way that search engines can search on related words as well as exact words, Oracle can perform fuzzy matching on the street names and so on. So, for example, suppose the address field in the preceding example was misspelled as `'CONNETICUT AVE'`. The geocoder could perform approximate matching to match the misspelled fields with those in the database.

Note that the `SDO_GEOMETRY` data type is just like any other object type in the database. Users can view the data, and examine and modify the attributes. In contrast, several GIS data vendors and partners have their own proprietary binary formats for representing spatial information. These vendors usually provide tools for loading the data or converting the data into standard Oracle formats. Discussion of these tools, however, is beyond the scope of this book.

Query and Analysis

Now that you've seen how to define `SDO_GEOMETRY` for storage of spatial data in Oracle, and how to populate Spatial tables with data, the next thing to look at is how to query and analyze this `SDO_GEOMETRY` data.

The query and analysis component provides the core functionality for querying and analyzing spatial geometries. This component has two subcomponents: a Geometry Engine and an Index Engine. It is via these components that we perform our spatial queries and analysis, for example, to identify the five nearest restaurants along Interstate 795 or the five nearest hospitals to a construction site.

The Geometry Engine

The Geometry Engine provides functions to analyze, compare, and manipulate geometries. For instance, you could use the Geometry Engine functionality to identify the nearest five

restaurants on I-795 in the greater Washington, D.C. area. This involves computing the distance between I-795 and all the restaurants in the us_restaurants table, sorting them in order of increasing distance, and returning the top five restaurants. The SQL in Listing 2-4 illustrates this operation.

Listing 2-4. *Finding the Five Nearest Restaurants on I-795*

```
SQL> SELECT poi_name
FROM
  (
    SELECT poi_name,
      SDO_GEOM.SDO_DISTANCE(P.location, I.geom, 0.5) distance
    FROM us_interstates  I, us_restaurants  P
    WHERE I.interstate = 'I795'
      ORDER BY distance
  )
WHERE ROWNUM <= 5;

POI_NAME
----------------------------------
PIZZA BOLI'S
BLAIR MANSION INN DINNER THEATER
KFC
CHINA HUT
PIZZA HUT

5 rows selected.
```

Observe that the inner SELECT clause computes the distance between I-795 (which is not a major highway) and each "restaurant" row of the us_restaurants table using the Geometry Engine function SDO_GEOM.SDO_DISTANCE. Also, note that the ORDER BY clause sorts the results in ascending order of distance. The outer SELECT statement selects the first five rows, or the five nearest restaurants.

In the preceding query, the location of the I-795 highway is compared with every restaurant row of the table, irrespective of how far they are from I-795. This could mean considerable time is spent in processing rows for restaurants that are too far from the I-795 highway and hence are irrelevant to the query. To speed up query processing by minimizing the processing overhead, we need to create *indexes* on the location of the restaurants.

The Index Engine

Oracle Spatial provides the spatial Index Engine for this purpose. Listing 2-5 shows an example of how to create an index on the locations of restaurants.

Listing 2-5. *Creating an Index on Locations (*SDO_GEOMETRY *Column) of Restaurants*

```
SQL> DROP INDEX us_restaurants_sidx;
SQL> CREATE INDEX us_restaurants_sidx ON us_restaurants(location)
INDEXTYPE IS mdsys.spatial_index;
```

Listing 2-5 first drops the index that exists. In the second and third lines, it shows the SQL for creating the spatial index. Note that the clause INDEXTYPE tells the database to create a spatial index on the location (SDO_GEOMETRY) column of the us_restaurants table. This index is a specialized index to cater to the SDO_GEOMETRY data. Using such an index, the Index Engine in Oracle Spatial prunes far-away rows from query processing and thus speeds up the query for most applications. The Index Engine provides equivalent functions, referred to as *operators*, for identifying rows of the table that satisfy a specified proximity predicate such as closeness to I-795. You can rewrite the preceding query to find the five nearest restaurants to I-795 using such *index-based operators*. Listing 2-6 shows the resulting query.

Listing 2-6. *Finding the Five Nearest Restaurants on I-795 Using the Spatial Index*

```
SQL> SELECT  poi_name
FROM us_interstates  I,  us_restaurants  P
WHERE I.interstate = 'I795'
  AND SDO_NN(P.location, I.geom) ='TRUE'
  AND ROWNUM <= 5;
POI_NAME
----------------------------------
PIZZA BOLI'S
BLAIR MANSION INN DINNER THEATER
KFC
CHINA HUT
PIZZA HUT

5 rows selected.
```

Note that this query returns the same five rows as Listing 2-4. However, this query has a simpler structure with no subqueries. It uses only a new index-based operator called SDO_NN, with NN being short for Nearest-Neighbor. This index-based operator returns rows of the us_restaurants table whenever the location column is close to the I-795 highway geometry. The SDO_NN operator returns these rows in order of proximity to the I-795 geometry. So, the row with closest location is returned first, the next closest next, and so on. The ROWNUM predicate determines how many close restaurants need to be returned in the query. The query uses a spatial index and examines only those rows that are likely to be close to the location of I-795. Consequently, it is likely to execute faster than the query in Listing 2-4.

As a variation on this, suppose that instead of having to find the five nearest restaurants on I-795, you wish to identify all restaurants within 50 kilometers of I-795. One way to accomplish this would be to construct a buffer around the I-795 highway and determine all businesses

inside this buffer geometry. Figure 2-2 shows an example. I-795 is shown in black. The 50 km buffer is shown by the gray oval around it, and the restaurants inside this buffer are shown by *x* marks.

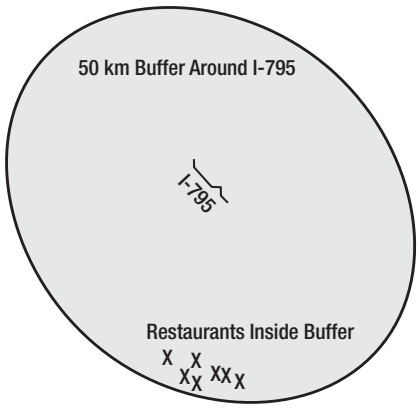

Figure 2-2. *Restaurants in the 50 km buffer around I-795*

Listing 2-7 shows the corresponding SQL query and the results.

Listing 2-7. *Identifying All Restaurants in a 50 km Radius Around I-795*

```
SQL> SELECT POI_NAME
FROM us_interstates I,  us_restaurants P
WHERE
  SDO_ANYINTERACT
  (
    P.location,
    SDO_GEOM.SDO_BUFFER(I.geom, 50, 0.5, 'UNIT=KM')
  ) ='TRUE'
  AND I.interstate='I795' ;
POI_NAME
----------------------------------
SPICY DELIGHT
PHILLY'S STEAK EXPRESS
EL TAMARINDO
MCDONALD'S
PIZZA HUT
CHINA HUT
KFC
BLAIR MANSION INN DINNER THEATER
PIZZA BOLI'S

9 rows selected.
```

The function SDO_ANYINTERACT is an index-based operator just like the SDO_NN operator in Listing 2-6. This operator identifies all rows of us_restaurants where the locations intersect with the geometry passed in as the second parameter. The second parameter, in this case, is the result returned by an SDO_BUFFER function. The SDO_BUFFER function generates and returns a 50 km buffer around the I-795 geometry. This SDO_BUFFER function is part of the Geometry Engine, which also provides additional functions to facilitate more complex analysis and manipulation of spatial information.

Note that the number of restaurants returned in Listing 2-7 is nine, as opposed to five in Listings 2-4 and 2-6. This means that we may not know the cardinality of the result set when we use a query buffer. With an SDO_ANYINTERACT operator, we may get more answers than we expect, or fewer answers. The cardinality of the result set depends on distribution of the data (in other words, the restaurants). In general, when you know how far to search (for example, a 50 km radius, as in Listing 2-7), you may use the SDO_BUFFER and SDO_ANYINTERACT functions.[3] Alternatively, if you know how many results you wish to return, then you should use the SDO_NN function, as described in Listing 2-6. In Chapters 8 and 9, we will describe in greater detail the different operators and functions in the Index and Geometry Engines.

Visualizing Spatial Data

How do you visualize the results of spatial queries? Oracle technology includes the MapViewer component to facilitate generation of maps from spatial data. Each map is associated with a set of *themes*. Each theme denotes spatial data from a specific table and is associated with a *rendering style*. For instance, you can specify that the interstates theme (data from the INTERSTATES table) should be rendered as thick blue lines. Oracle Spatial provides appropriate dictionary views, USER_SDO_MAPS, USER_SDO_THEMES, and USER_SDO_STYLES, to define new maps, to associate them with themes, and to specify rendering styles for the themes inside the database, respectively.

In addition, MapViewer renders the map for a specified map name. Basically, a servlet consults the database views and retrieves the themes and associated styling rules for a specified map name. Using this information, the MapViewer servlet generates an image of the constructed map. Figure 2-3 shows an image of such a map constructed using MapViewer (constructed entirely using spatial technology and the data provided in this book). This map shows I-795 along with the larger interstates.

3. In Chapter 8, we will describe a better alternative using SDO_WITHIN_DISTANCE operator.

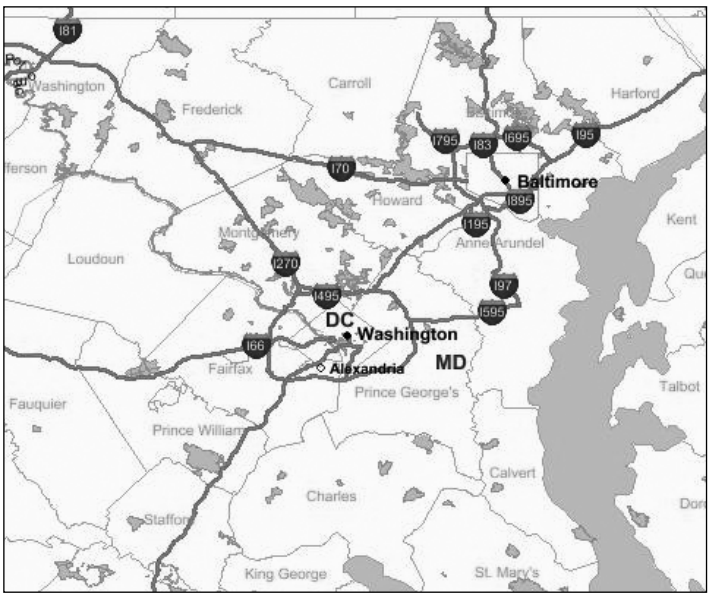

Figure 2-3. *Sample map with multiple themes generated using MapViewer*

The map consists of multiple themes: cities, county boundaries, rivers, interstates, and parks. The cities, D.C. and Baltimore, are rendered as points in black color. The counties, Howard, Fairfax, Charles, Frederick, etc., are shown as white polygons. The river in the right side of the map is shown in a dark grey color. The interstates, such as I-795, are rendered as *line strings* in black, and the parks are rendered as *polygons* in light gray.

Onto this map, we can also superimpose the locations of the five closest restaurants to I-795. In addition to rendering predefined themes/base maps, the MapViewer request can specify a predefined base-map (such as the map in Figure 2-3) and a *dynamic* theme, such as a SQL/JDBC query retrieving the locations of the five nearest restaurants. MapViewer will then generate a new map that contains the locations of the five restaurants superimposed on the predefined base map.

Note that the map in Figure 2-3 displays vector data stored as `SDO_GEOMETRY` columns in different (theme) tables. In addition to vector data, MapViewer can display spatial data stored in the raster (or image) format. Such data is stored in Oracle tables using the `SDO_GEORASTER` data type. Chapter 11 provides full details of how to construct maps and display the results of queries on such maps using MapViewer technology.

Advanced Spatial Engine

The Advanced Spatial Engine has several subcomponents that cater to the complex analysis and manipulation of spatial data that is required in traditional GIS applications.

Note Our focus in this book is the applicability of Oracle Spatial to Oracle business applications, so we do not cover most of these advanced options, with the exception of the network data model, in great detail. However, we provide a good overview of these topics in the appendixes, with references for further details.

Internally, each of these additional components uses the underlying geometry data type and index and geometry engine functionality.

- The *Network Data Model* provides a data model for storing networks inside the Oracle database. Network elements (links and nodes) can be associated with costs and limits, for example, to model speed limits for road segments. Other functionality includes computation of the shortest path between two locations given a network of road segments, finding the *N* nearest nodes, and so on. The network data model is useful in routing applications. Typical routing applications include web services such as MapQuest and Yahoo! Maps, or navigation applications for roaming users using GPS technology. We cover more details about this component in Chapter 10.
- The *Linear Referencing System* (LRS) facilitates the translation of mile-markers on a highway (or any other linear feature) to geographic coordinate space and vice versa. This component allows users to address different segments of a linear geometry, such as a highway, without actually referring to the coordinates of the segment. This functionality is useful in transportation and utility applications, such as gas pipeline management.
- The *Spatial Analysis and Mining Engine* provides basic functionality for combining demographic and spatial analysis. This functionality is useful in identifying prospective sites for starting new stores based on customer density and income. These tools can also be used to materialize the influence of the neighborhood, which in turn can be used in improving the efficacy and predictive power of the Oracle Data Mining Engine.
- *GeoRaster* facilitates the storage and retrieval of georeferenced images using their spatial footprints and the associated metadata. GeoRaster defines a new data type for storing raster images of geographically referenced objects. This functionality is useful in the management of satellite imagery.
- The *Topology Data Model* supports detailed analysis and manipulation of spatial geometry data using finer topological elements such as nodes and edges. In some land-management applications, geometries share boundaries, as in the case of a property boundary and the road on which the property is situated. Oracle Spatial defines a new data type to represent topological elements (such as the shared "road segment") that can be shared between different spatial objects. Updates to shared elements implicitly define updates to the sharing geometry objects. In general, this component allows for the editing and manipulation of nodes and edges without disturbing the topological semantics of the application.

Oracle Spatial Technology Products

In the previous sections, we briefly described the functionality that Oracle Spatial provides to support the following operations on spatial data:

- Storage data model using the SDO_GEOMETRY data type
- Query and analysis using the Index Engine and Geometry Engine
- Location enabling using the geocoder by conversion of address data into SDO_GEOMETRY data
- Visualization using MapViewer
- Advanced Spatial Engine functionality such as network analysis

Let's next look at how this functionality is productized or licensed in Oracle Database 10*g*, version 10.1.0.2, and Oracle Application Server 10*g*, version 9.0.4. Note, though, that this packaging may change with later versions of Oracle.

MapViewer, the visualization tool of Spatial, is included as part of the Oracle Application Server. You can also deploy MapViewer by just installing the Oracle Containers for Java (OC4J) without installing the entire Application Server. We will look at these details in Chapter 11. The remainder of the Spatial functionality is included, sometimes optionally, with the Database Server. Let's look at these details next.

In the Lite edition of Oracle Database Server, none of the spatial functionality is included. As mentioned in an earlier Note, in the Personal, Standard,[4] and Enterprise editions, a *subset* of the spatial functionality is included for free with the database. This subset is referred to as the *Locator*. In the Personal and the Enterprise Editions, the full functionality of spatial technology is available as a *priced* option, called *Spatial*. Let's look at each of these versions of Oracle Spatial and what you can do with them.

Locator

Locator provides a *core* subset of spatial functionality to cater to specific applications. Specifically, it includes the following functionality:

- *The data model for storing spatial data using the* SDO_GEOMETRY *data type*: This includes storing all types of geometries (points, lines, polygons, and so on).
- *Query and analysis using the Index Engine*: This includes creating spatial indexes and querying using associated spatial operators like SDO_NN.
- *The* SDO_GEOM.SDO_DISTANCE *and the* SDO_GEOM.VALIDATE_GEOMETRY_XXX *functions*: These functions are also part of Locator.

Figure 2-4 shows the functionality provided in Locator. The Locator components are highlighted in black. The non-Locator components of Spatial technology are shown in solid gray.

4. "Standard" implies both Standard Edition One and Standard Edition.

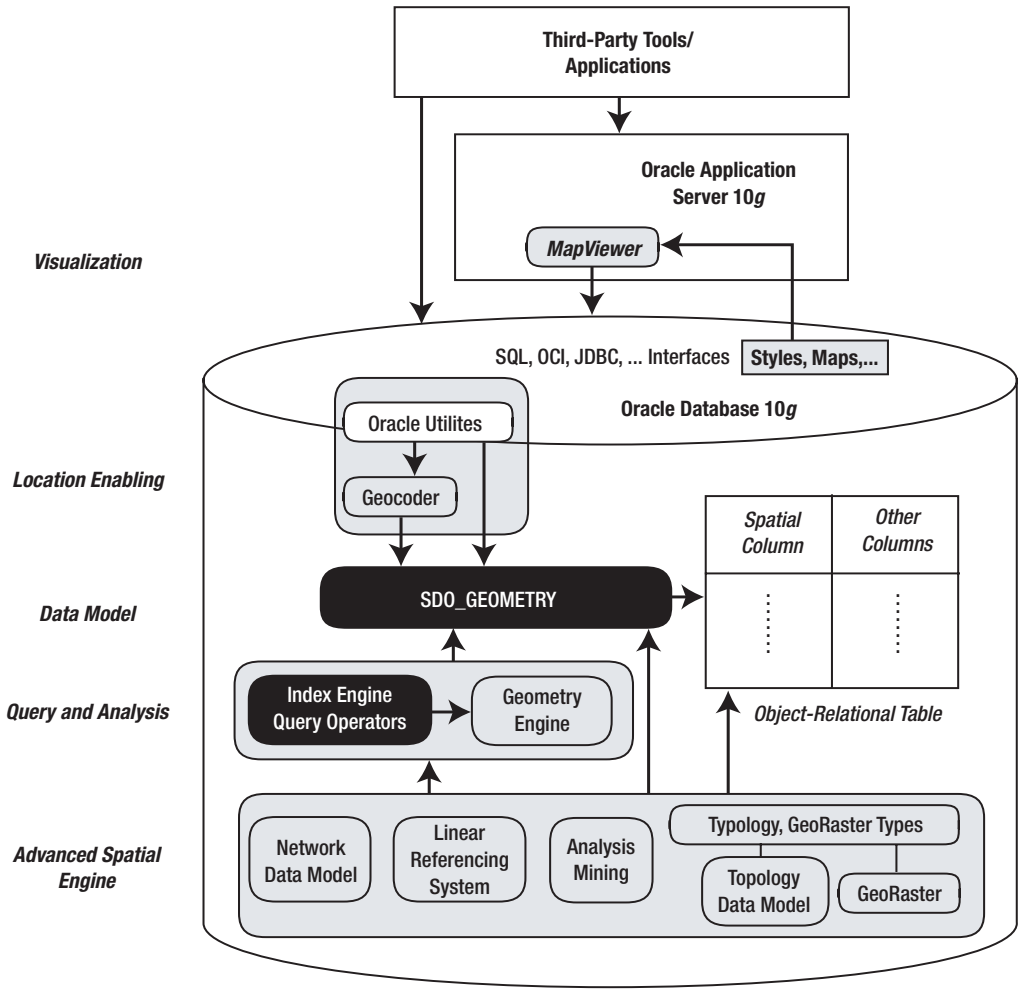

Figure 2-4. *The functionality of Locator, the free part of Spatial technology, is shown in black ovals.*

Applications that use Locator may need to use third-party geocoding services to convert addresses in application tables. After storing the spatial locations as `SDO_GEOMETRY` columns, Locator enables a variety of spatial queries, such as identification of customers within a specified sales territory or the nearest ATM to a specific location. Locator is typically used in the following applications:

- *Simple GIS applications*, which may just work with geographic data such as state, city, or property boundaries and index-based query using associated spatial operators. Typically, though, most GIS applications may need the Geometry Engine functionality (which is not supported in Locator).
- *Simple business applications*, where the spatial data is obtained from third-party vendors. As you will see in Chapter 8, the index-based operators supported in Locator can perform a great deal of analysis in business applications.

- *CAD/CAM and similar applications*, where the spatial data does not refer to locations on the surface of the Earth. For instance, in CAD/CAM applications, the data represents the structure/shapes of different parts of an automobile. In this case, the data is inherently in the two- or three-dimensional coordinate space—that is, there is no need to convert nonspatial columns (such as addresses) to obtain spatial information. The operations that are needed for such applications are the index-based proximity analysis operators. The advanced spatial engine functions such as routing are of no use in these applications.

To summarize, Locator offers a *core subset* of spatial technology. If you want to exploit the full feature-set of spatial technology, you will need to purchase the Spatial option in the Enterprise Edition of Oracle Database.

Spatial Option

The Spatial option is a priced option of the Enterprise Edition of Oracle Database Server. This option includes all the components of the spatial technology referred to in Figure 2-4 and is a superset of Locator. Figure 2-5 shows the functionality of the Spatial option in gray. Note that the Spatial option does not include the MapViewer component (shown in black) of spatial technology. The Spatial option consists of

- *Storage data model using* `SDO_GEOMETRY` *data type*: This includes storing of all types of geometries (points, lines, polygons, and so on).
- *Query and analysis using the Index Engine*: This includes creating spatial indexes and querying using associated spatial operators like `SDO_NN`.
- *Query and analysis using the Geometry Engine*: This supports different analysis functions for individual geometries, pairs of geometries, or a set of geometries.
- *Location enabling using the geocoder*: This facilitates conversion of address data into `SDO_GEOMETRY` data.
- *Advanced Spatial Engine functionality*: This includes routing and network analysis.

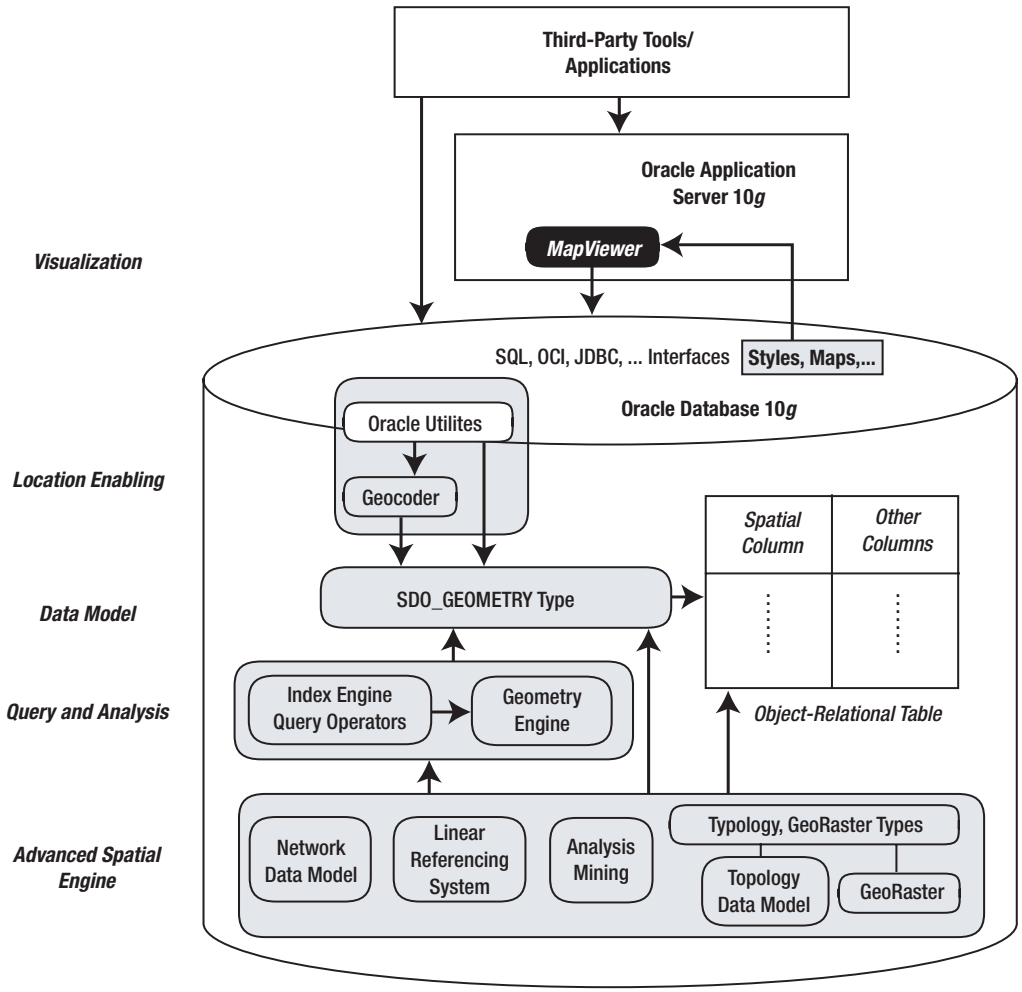

Figure 2-5. *The functionality of the Spatial option is shown in gray.*

A wide variety of applications can use the full set of functionality provided in the Spatial option.

By now, you should have a good idea of how Oracle Spatial functionality is packaged. This understanding is helpful in determining whether or not your application needs to license the full set of spatial functionality using the Spatial option. For the remainder of this book, we will not differentiate or explicitly refer to Locator and Spatial option products. Instead, we will refer to the entire set of functionality as "Oracle Spatial technology" or simply as "Oracle Spatial."

What to Expect in an Oracle Spatial Install

In this section, we discuss what to expect during or after you install Oracle Spatial technology inside the Oracle Database Server. We describe how to install the MapViewer component, which is part of Oracle Application Server 10*g*, separately in Chapter 11.

Installing Oracle Spatial in the Database

As noted previously, Oracle Spatial is automatically installed with the Standard or Enterprise Edition of an Oracle Database Server. All Spatial data types, views, packages, and functions are installed as part of a schema called `MDSYS`.

To verify that Spatial has been installed properly, you first have to check that the `MDSYS` account exists. If it does not, then Spatial is not installed. Otherwise, you can execute the SQL in Listing 2-8 after connecting through your `SYS (SYSDBA)` account.

Listing 2-8. *Verifying That a Spatial Install Is Successful*

```
SQL> SELECT COMP_NAME, STATUS
FROM DBA_REGISTRY
WHERE COMP_NAME = 'Spatial';
COMP_NAME                      STATUS
------------------------------ -----------
Spatial                        VALID
```

After a successful installation, the status will be set to `VALID` or `LOADED`.

Upgrades

To understand upgrades properly, let's look at how Spatial technology evolved between different versions of Oracle. Figure 2-6 shows the progression from Oracle 7.2 to Oracle 10*g*.

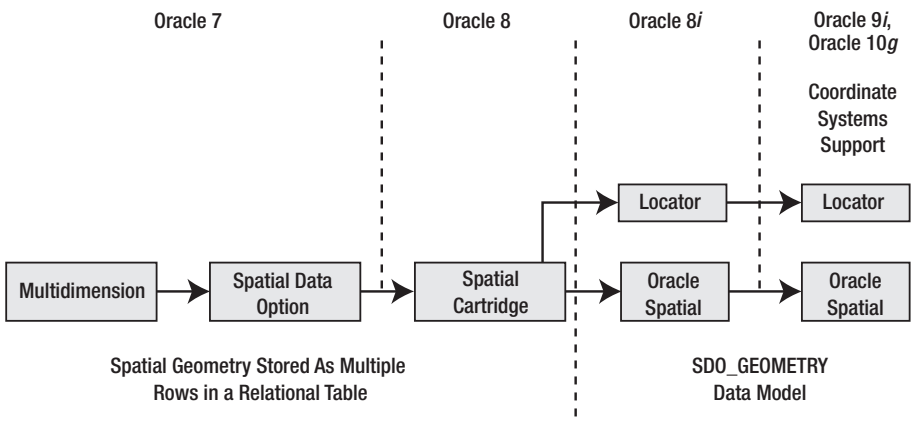

Figure 2-6. *Evolution of Spatial technology in Oracle*

Spatial technology was first introduced in Oracle 7.2 under the name *Oracle MultiDimension* (MD). Later, the product name was changed to *Oracle Spatial Data Option* (or SDO) and to *Spatial Data Cartridge* in Oracle 8. Since objects were not supported in these releases, the coordinates of a geometry were stored as multiple rows in an associated table. Managing spatial (geometry) data in these prior versions was inefficient and cumbersome.

Starting with Oracle8*i*, the `SDO_GEOMETRY` data type was introduced to store spatial data. Even in Oracle 10*g*, the same `SDO_GEOMETRY` model is used to store spatial data in Oracle. In Oracle9*i* (and Oracle 10*g*), the geometry data also included support for coordinate systems information specified using the `SRID` attribute in the `SDO_GEOMETRY` data type. In Oracle 10*g*, additional functionality (that exists in the Advanced Spatial Engine) such as the Network Data Model is introduced in the Spatial option of Oracle.

Since the prior versions are named MultiDimension (MD) and Spatial Data Option (SDO), you will see the prefixes `MD` and `SDO` for the files and schemas that install Spatial technology. The name of the spatial install schema is `MDSYS` in all versions of Oracle.

In spite of the evolution of Spatial technology with each release, upgrading to the latest version, Oracle 10*g*, is not difficult. *Spatial technology is automatically upgraded with the upgrade of Oracle Database Server.* The upgrade may not need your intervention at all.[5] However, if you are upgrading from pre-8*i* releases, you need to additionally migrate your geometry data from the pre-8*i* format to the `SDO_GEOMETRY` data model. Oracle Spatial provides the `SDO_MIGRATE` package to migrate the data from pre-8*i* models to the current `SDO_GEOMETRY` data model. We discuss this migration package's functionality in Chapter 5.

Understanding a Spatial Install

In this section, we cover where to find appropriate spatial files and how to perform some preliminary investigation when an installation or upgrade fails.

To view all the spatial files, you can go to the `$ORACLE_HOME/md/admin` directory. In this directory, you will find all files relevant to Oracle Spatial. You will observe that a majority of the files have a prefix of either `SDO` or `PRVT`. In other words, the files are of the form `sdoxxxx.sql` or `prvtxxxx.plb`. The `SDO` files, in most cases, contain package definitions for different components of Spatial technology. The `PRVT` files, on the other hand, are binary files and define the package bodies and so on.[6] You should not tamper with these `SDO` and `PRVT` files at any time.

During the creation of the database,[7] the `MDSYS` account is created with appropriate privileges (see scripts `mdinst.sql` and `mdprivs.sql` for more details) and the `catmd.sql` file is loaded into the `MDSYS` schema. This file loads all the `SDO` and `PRVT` files in an appropriate order that resolves all dependencies between all the Spatial packages. In the case of Locator, `catmdloc.sql` (instead of `catmd.sql`) is loaded. Likewise, appropriate files in this directory such as `sdodbmig.sql` or `sdopatch.sql` are loaded/executed at the time of upgrades, downgrades, and patches.

---

5. Note that some spatial components such as GeoRaster have dependencies on other Oracle components such as interMedia and XML. You need to ensure that these components are also upgraded properly or installed if they do not exist in a custom install.
6. Most functions in these package bodies are linked to C/Java libraries that are included with the Oracle kernel.
7. The database can be created either at install time or using a variety of Oracle tools such as DBCA.

During some installations or upgrades, you may find that several package dependencies are unresolved and hence invalid. You may check for such invalid packages or other objects in your Spatial installation by running the SQL in Listing 2-9.

Listing 2-9. *Checking for Invalid Objects in a Spatial Installation*

```
SQL> SELECT OBJECT_NAME, OBJECT_TYPE, STATUS
FROM ALL_OBJECTS
WHERE OWNER='MDSYS'  AND STATUS <> 'VALID'
ORDER BY OBJECT_NAME;
```

If Listing 2-9 returns any rows, you should contact Oracle Support for troubleshooting help.

Summary

This chapter provided a brief overview of the various components of Oracle Spatial technology. First, we examined the functionality provided in Oracle Spatial. This functionality included a SQL-level data type for storing spatial data, new operators and functions to perform spatial query and analysis, a MapViewer tool for visualizing spatial data, and advanced components to perform more sophisticated analysis such as routing or network analysis. We then described how this functionality is packaged in the Database and Application Servers. Finally, we described what to expect in a typical Spatial installation and where to find appropriate Spatial files.

Starting with the next chapter, we will look at Oracle Spatial functionality in more detail. Specifically in Chapter 3, we describe how to location-enable your application.