Oracle Database 10g

Developing Spatial Applications
Using Oracle Spatial and MapViewer

An Oracle Technical White Paper
February 2004
# TABLE OF CONTENTS

1 INTRODUCTION ...........................................................................................................1

2 ORACLE SPATIAL ........................................................................................................2

3 ORACLE APPLICATION SERVER MAPVIEWER .............................................................2

4 CASE STUDY: SHIP TRACKING .................................................................................5

5 SUMMARY .........................................................................................................................9

6 WEB RESOURCES.........................................................................................................10
1 Introduction
This white paper introduces briefly the main features of Oracle Spatial 10g and Oracle Application Server MapViewer 10g. Through a case study we also show that these off-the-shelf products can serve as effective building blocks for large-scale spatiotemporal data analysis and knowledge discovery projects or applications.

2 Oracle Spatial
Oracle Spatial is part of the Enterprise Edition of Oracle Database 10g. It is a foundation for the deployment of enterprise-wide spatial information systems, and Web-based and wireless location-based applications requiring complex spatial data management. It provides a rich set of natively supported open standard data types and models, as well as a coherent and high performant SQL interface for accessing and managing such data.

In the 10g version, Oracle Spatial natively supports the following data types and models:

- Vector data: all OGC simple feature geometry types and essential operators on them, including relate and aggregation. Supported since version 8.1.6, this object-relational vector data model has been greatly enhanced in many aspects, including support for spatial reference/coordinate systems and linear referencing system.

- Topology and Network data models: Oracle Spatial 10g natively supports topology and network data models. For example, it can manage the node/edge/face information of a topology geometry layer, and also provides a comprehensive set of functions to operate on the topology data. Based on the topology data model, Oracle Spatial also implements a network data model which manages such logical information as connectivity relationships among nodes and links, directions of links, and costs of nodes and links. Many network applications from biochemical areas to transportations can already be developed based on such information and built-in network algorithms in Spatial.

- GeoRaster. Oracle Spatial 10g lets you store, index, query, analyze, and deliver georaster data, that is, raster image data and its associated Spatial vector geometry data, plus metadata. GeoRaster provides Oracle Spatial data types and an object-relational schema for storing multidimensional grid layers and digital images that can be referenced to positions on the Earth's surface or a local coordinate system.

Another significant new capability added to Spatial 10g is the Spatial analysis and mining (SAM) package. With the SAM functions, user can effectively use the general purpose Oracle Data Mining engine while taking into account the characteristics of spatial data, such as spatial auto-correlation. SAM currently supports such functions as spatial binning, proximity and co-location analysis, as well as spatial clustering.

The fact that these spatial data types and functions are provided in the context of a general purpose database that also supports other rich data types (including multimedia
and XML) and a vast array of management and analysis functions can never be over emphasized. It is the convergence of such diverse array of data types, data models and processing functions in a single environment that makes many types of complex applications possible, as will be illustrated through the example in a later chapter. It also means that spatial applications built on top of Oracle Spatial can benefit directly from the scalability and high performance characteristics and mechanisms of Oracle database.

Oracle Spatial 10g also comes with a new set of Java spatial access APIs, which you can use to customize or extend the functionalities already provided by Oracle Spatial. For example, generating Thiessen polygons (or Voroni diagram) is not an existing function of Oracle Spatial. However, if you are proficient with Java, it is quite easy to write your own function that implements the Voroni diagram generation algorithm. The function’s input can be as simple as a query that selects a set of points from an existing table in the database. Basically, your function first executes the query using the server-side JDBC driver and then uses the Java spatial API to access the selected spatial objects (points in this case), before actually generating the Thiessen polygons. Once you have finished coding and debugging your function outside the database, you can then load it into the database server as a stored function, which can be invoked from any tool that can issue SQL commands to the database by other users. You can also code the function so that it writes the generated polygons into a new table, which can then be visualized by a wide range of tools, including MapViewer.

It is also worth noting that the Oracle Spatial vector data model is widely supported by GIS vendors’ software; support for the new data types and models in 10g has also been pledged by most leading GIS vendors and spatial/LBS software developers. The openness of Spatial’s data types and models is making it a de facto industry standard.

3 Oracle Application Server MapViewer

MapViewer is a programmable tool for visualizing spatial data managed by Oracle Spatial. It is basically a Java servlet that runs inside a J2EE (Java Enterprise Edition) container, in this case Oracle’s Application Server. Once up and running, it acts as a map server that awaits user’s map request through HTTP, and sends back a map response after processing and generating a map. The maps generated by MapViewer are highly customizable, with all the mapping metadata such as map symbols and styling rules stored in the database and managed through a graphical user interface. MapViewer works directly with Oracle Spatial through Java JDBC.

MapViewer also supports thematic mapping where styles can be dynamically created based on the result of knowledge discovery queries. For example, you may run a Oracle Discoverer report on a large volume sales data table, and based on the aggregated sales numbers range, dynamically create a color-series style to plot each sales region in a certain color based on its associated sales number.

The map request and response are always encoded in XML. This makes it very versatile in that any computer language that can send data through HTTP is able to issue map requests to MapViewer. For example, you can write a PL/SQL code (an Oracle
procedural query language that runs inside the database engine) to construct a dynamic map request, send it outside the database to the MapViewer server, retrieve the generated image and save it in a database table. Later we will see an example where this can be useful in an application for detecting spatial outlier.

Working directly with XML, however, can be cumbersome and error-prone in most cases. Mapviewer also provides a Java client API, which can be used to construct map requests from a Java application or servlet, or JSP (Java Server Page).

MapViewer and Oracle Spatial combined can also serve as a powerful GIS educational platform, in that students can both directly work on a modern spatial database and immediately visualize the work they have done in the database. For example, Figure 1 displays a SQL-based buffer operation and its textual result, and Figure 2 shows the same query result displayed through MapViewer.

![Figure 1 SQL Access to Geometries](image-url)

Figure 1 SQL Access to Geometries
Figure 2 Visualizing Spatial Query Results Using MapViewer

Figure 3 shows a more complex map generated by MapViewer. It is part of an experimental application that uses the geocoding capabilities of Oracle Spatial. The application loads the street data for the entire U.S. into Oracle Spatial, and sets up a Web front end that provides mapping, geocoding, and routing services through a simple XML API. The mapping service is based on MapViewer, the geocoding service is based on the Spatial geocoding package, and the routing service is based on the Spatial network model.
Figure 3 Sample LBS Application Developed Using Spatial and MapViewer

4 Case Study: Ship Tracking

Let's now focus on an imaginative knowledge discovery and event notification system. In this test case the main purpose is real time tracking of shipping liners' locations as they move through canals or oceans, and to notify concerned parties of any abnormal behavior or outlier.

Assume that before each ship departs, its normal or planned course to the destination can be entered into a database table. We can have the following table schema:
TABLE ship_planned_routes (ship_id number(10),
destination varchar2(256),
route SDO_GEOMETRY)

Note that in the above definition the “route” object is defined by an Oracle data type SDO_GEOMETRY, which is the object type for representing all kinds of geometries. The route can be generated from past ship courses with the same departure point and destination. To allow a reasonable amount of deviation, we probably should create a buffer around the route (which is essentially a line string) and store the buffered route (a polygon) in the above table.

Immediately after a ship departs, we can start tracking its locations at a fixed interval (e.g., every few seconds), and in real-time inserting the records into a table with the following simplified schema:

TABLE ship_locations (ship_id number(10),
time timestamp(3),
location SDO_GEOMETRY)

Note that in this table the time is represented using Oracle’s SQL type TIMESTAMP, which is a high-precision time and date type. The numeric value 3 here means we want Oracle to use 3 digits when storing the fractions of a second. The timestamp values are basically points on a linear time axis.

We can imagine that whenever a ship’s GPS receiver updates its location, an on-board transmitter will automatically send out the current timestamp and location of the ship via satellite link to a land-based monitoring station, where the information is inserted into the above table through a query like the following:

```
INSERT INTO ship_locations VALUES(1234,
to_timestamp('11-NOV-2003 14:01:13.00', 'dd-mon-yyyy hh24:mi:ss.ff'),
mdsys.sdo_geometry(2001, 8265,
mdsys.sdo_point_type(-140.2491, 32.8835,null),
null, null));
```

The above query inserts a new record for ship 1234’s location (longitude -140.2491, latitude 32.8836) at the specified timestamp (November 11, 2003, 14:01:13). It is possible with Oracle Spatial to insert thousands of such records in one second, which makes it quite feasible for real time tracking of hundreds of ships or other types of moving objects.

Now, all we need to do is writing a database trigger that implements the monitoring logic. A database trigger is basically a stored procedure that will get fired whenever a new record is inserted into a specific table. In the trigger body, we can simply check for any deviation of the current ship location from its planned ship route. This operation can be achieved using Oracle Spatial’s relate function. Remember that the planned route is
stored as a buffered zone around the actual route. We can now issue the following sample query from the trigger body to determine if the ship is still within the planned route’s buffer:

```sql
SELECT sdo_geom.relate(a.location, 'ANYINTERACT', b.route, 0.5)
FROM ship_locations a, ship_planned_routes b
WHERE a.ship_id = b.ship_id and a.time =
    (SELECT max(time) from ship_locations
     where ship_id=1234) and a.ship_id=1234;
```

The result of this query, which is actually the result of the Spatial function SDO_GEOM.RELATE, will be ‘TRUE’ if the ship is still within the planned route, or ‘FALSE’ otherwise. Note that the sub-query (select max(time) from ship_locations where ship_id=1234) will return the most current timestamp for the ship with id 1234. If a ‘FALSE’ value is returned, the trigger body can, for example, open an HTTP connection to a “dispatcher” Web service outside the database, and send a message to the dispatcher with the ship id, the location and the time of deviation. The dispatcher service can then act accordingly, such as sending out emails to all concerned parties using Java Mail, and/or highlighting the deviating ship’s planned route and current location on a dynamic map by issuing a map request to a MapViewer server.

The real-time ship movement data as well as planned ship routes can all be plotted on a map using MapViewer. The map can be refreshed according to a specified interval to update the ship locations for visual monitoring. This can be achieved by periodically sending a map request to the MapViewer server from an auto-refresh Web page. The XML map request will look something like the following:

```xml
<?xml version="1.0" standalone="yes"?>
<map_request
title="Ship Tracking Map"
basemap="world_map"
datasource = "ship_db"
>
<themes>
  <theme name="ship_locations_in_last_30_min">
    <jdbc_query
      spatial_column="location"
      render_style="C.RED"
    >
      SELECT geom from ship_locations
      where time > systimestamp - to_dsinterval('0 00:30:00')
    </jdbc_query>
  </theme>
  <theme name="planned_routes">
    <jdbc_query
      spatial_column="route"
      render_style="C.BLACK"
      label_column="ship_id"
    >
```
Note that in the above simplified MapViewer request, we are instructing MapViewer to create and render two dynamic themes or layers based on the result of the supplied SQL query. The first theme is basically selecting all the locations for all the ships that are recorded within the past 30 minutes. The second theme is simply selecting all the planned routes from the ship_planned_routes table, each route being annotated with its ship id. Such a map can provide a bird’s eye view of all the currently active ships’ routes and locations, with any outlier automatically detected and notifications automatically sent out by the system.

To make our sample application a bit more sophisticated, we can also introduce some predefined spatiotemporal zones to validate not only the spatial proximity, but also the temporal proximity. For example, we may want to establish a set of “must-arrive” zones along the planned route for a given ship. Such a zone will demand that a ship not only passes through it, but that the time it arrives at the zone is according to a preset schedule or interval. In this case, we can simply use Oracle’s object-relational capabilities to create user-defined object types that represent such events or spatiotemporal zones. Below is such a sample type named SPATIOTEMPORAL_ZONE:

```sql
Create or replace type Spatiotemporal_Zone
as object
(name varchar2(64),
earliest timestamp(3),
latest timestamp(3),
area SDO_GEOMETRY);
```

This type is probably the simplest form of user-defined objects. To make it more useful we can, for example, add a method to it that validates if a ship’s location and time of arrival falls inside a zone. With such a user object type defined in our system, we can then create a table that stores all the zones along all the planned shipping routes, and have the database trigger that we defined previously call a corresponding zone’s method to validate the ship movement. Without much stretch of the imagination, we can also define other user object types in the database that can capture and manage large volumes of histories and changes happening to real world objects. The main impedance here is the lack of an industry standard regarding spatiotemporal data semantics and operations.

Figure 4 shows one of the many maps dynamically generated by MapViewer while tracking the ship with id 1234 using a simulated set of data. The long gray polygon is a 100-mile buffer around the planned route for the ship. The big red circle denotes an instance of the user defined type spatiotemporal_zone, which indicates an intermediate
stop that the ship must cross at the right time and within the right boundary. The small
red dots and the ship symbol denotes the positions of the ship in the past few timestamps
as well as the most current one. The underlying country data is stored in a separate table.

Figure 4 Map Tracking a Moving Ship

5 Summary

In this white paper we briefly introduced the main (new) features of Oracle Spatial 10g
and MapViewer. Through the case study, we have demonstrated that the integrated
 functionalities provided in Oracle Spatial, MapViewer, and the database server and
application server in general can serve as a powerful environment for building research
prototypes and real-world large-scale spatiotemporal applications. The future direction
for Oracle Spatial and MapViewer will include better compliance with open standards,
better usability in managing and visualizing geospatial data, as well as more analysis and
KD functions and modeling capabilities.
6 Web resources
