# Oracle Spatial: Essentials

**Volume I • Student Guide**

D56709GC10
Edition 1.0
January 2009
D57732

ORACLE

**Authors**

Daniel Geringer

Puja Singh

**Technical Contributors and Reviewers**

Daniel Abugov

Albert Godfrind

Daniel Geringer

Steve Friedberg

Harry Penberthy

Yash Jain

Thomas Hoogerwerf

Nancy Greenberg

Maria Billings

Christopher Wensley

**Editors**

Nita Pavitran

Aju Kumar

Amitha Narayan

Vijayalakshmi Narasimhan

**Graphic Designer**

Satish Bettegowda

**Publishers**

Michael Sebastian Almeida

Jobi Varghese

Giri Venugopal

# Contents

# Preface

## Profile

### Before You Begin This Course

Before you begin this course, you should know how to use the Structure Query Language (SQL). You should be familiar with data processing concepts and techniques. You should also be familiar with object relational data model and the fundamentals of mathematical geometry concepts.

### How This Course Is Organized

***Oracle Spatial: Essentials*** is an instructor-led course featuring lectures and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills.

## Related Publications

### Oracle Publications

| Title | Part Number |
|---|---|
| *Oracle Spatial Developer's Guide 11g Release 1 (11.1)* | B28400-02 |
| *Oracle Application Server MapViewer User's Guide Release 10.1.3.1* | B14036-03 |

### Additional Publications

- System release bulletins
- Installation and user's guides
- *read.me* files
- International Oracle User's Group (IOUG) articles
- *Oracle Magazine*

## Typographic Conventions

The following table lists the typographical conventions that are used in text and code.

**Typographic Conventions in Text**

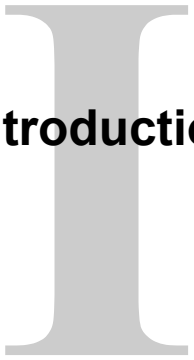
| Convention | Object or Term | Example |
|---|---|---|
| Uppercase | Commands, functions, column names, table names, PL/SQL objects, schemas | Use the `SELECT` command to view information stored in the `LAST_NAME` column of the `EMPLOYEES` table. |
| Lowercase, italic | Filenames, syntax variables, usernames, passwords | **where:** *role* is the name of the role to be created. |
| Initial cap | Trigger and button names | Assign a When-Validate-Item trigger to the ORD block.<br>Select Cancel. |
| Italic | Books, names of courses and manuals, and emphasized words or phrases | For more information on the subject see *Oracle SQL Reference Manual*<br>Do *not* save changes to the database. |
| Quotation marks | Lesson module titles referenced within a course | This subject is covered in Lesson 3, "Working with Objects." |

## Typographic Conventions (continued)

### Typographic Conventions in Code

| Convention | Object or Term | Example |
|---|---|---|
| Uppercase | Commands, functions | `SELECT employee_id`<br>`FROM  employees;` |
| Lowercase, italic | Syntax variables | `CREATE ROLE role;` |
| Initial cap | Forms triggers | `Form module: ORD`<br>`Trigger level: S_ITEM.QUANTITY item`<br>`Trigger name: When-Validate-Item`<br>`. . .` |
| Lowercase | Column names, table names, filenames, PL/SQL objects | `. . .`<br>`OG_ACTIVATE_LAYER`<br>`(OG_GET_LAYER ('prod_pie_layer'))`<br>`. . .`<br>`SELECT last_name`<br>`FROM  employees;` |
| Bold | Text that must be entered by a user | `CREATE USER scott`<br>`IDENTIFIED BY tiger;` |

# Introduction



# Objectives

After completing this lesson, you should be able to do the following:

- Explain the goals of the course
- Describe the features and application of the Oracle Spatial technology
- Discuss what is included in Oracle Locator and Oracle Spatial
- Review the object-relational data model
- Describe the classroom environment and the example data set used in the course

ORACLE



## Objectives

In this lesson, you are introduced to the Oracle Spatial or Locator technology history and package options. It briefly describes Oracle Locator (a database feature included in every edition of Oracle Database at no extra cost) and the features of Oracle Spatial (a licensed option).

This lesson briefly reviews the basic concepts and terminologies of the object-relational data model so that you become familiar with Oracle's way of implementing object types and methods.

You are introduced to the organization of the course and the classroom environment used in this course.

# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Spatial
- Overview of the object-relational data model
- Review of geometrical and geographical concepts
- The classroom environment and the example data set
- Oracle Spatial documentation and additional resources

ORACLE

# Course Objectives

After completing this course, you should be able to:

- Describe the Oracle Spatial data and query models
- Create spatial layers by using the `SDO_GEOMETRY` data type
- Load geometries into spatial layers
- Employ spatial operators and functions to generate and access 2D geometries
- Describe the various types of coordinate systems
- Run spatial queries to perform spatial analysis
- Enhance and tune spatial indexes for better performance
- Describe the linear referencing system
- Describe Oracle Spatial geocoding and routing concepts
- Use MapViewer and the Map Builder tool to render maps

ORACLE



## Course Objectives

This course introduces you to the Oracle Spatial technology. In this course, you learn about the fundamentals of modeling, storing, and querying spatial data.

The course covers the concepts and usage of the native data types, functions, and operators available in Oracle Database for spatial data. Using the Oracle Application Server MapViewer, you learn to render maps and view geospatial data managed by Oracle Spatial or Locator. You are also introduced to the basics of geocoding and routing concepts.

# Course Agenda

- Day 1:
  - Introduction
  - Lesson 1: Overview of Oracle Spatial Concepts
  - Lesson 2: Creating Spatial Layers
  - Lesson 3: Defining Collection Geometries
  - Lesson 4: Associating Spatial Layers with Coordinate Systems
- Day 2:
  - Lesson 5: Loading Spatial Data
  - Lesson 6: Validating and Debugging Geometries
  - Lesson 7: Using the Oracle Application Server MapViewer
  - Lesson 8: Indexing Spatial Data

ORACLE

# Course Agenda

- Day 3:
  - Lesson 9: Querying Spatial Data
  - Lesson 10: Using the `SDO_WITHIN_DISTANCE`, `SDO_NN`, and `SDO_JOIN` Operators
  - Lesson 11: Analyzing Geometries by Using Spatial Operators and Functions
  - Lesson 12: Using the Spatial Analysis, MBR, Utility, and Aggregate Functions
- Day 4:
  - Lesson 13: Defining Maps by Using the Map Builder Tool
  - Lesson 14: Leveraging Oracle Maps: The Map Cache and JavaScript API
  - Lesson 15: Creating a User-Defined Coordinate System
  - Lesson 16: Implementing a Linear Referencing System

ORACLE

# Course Agenda

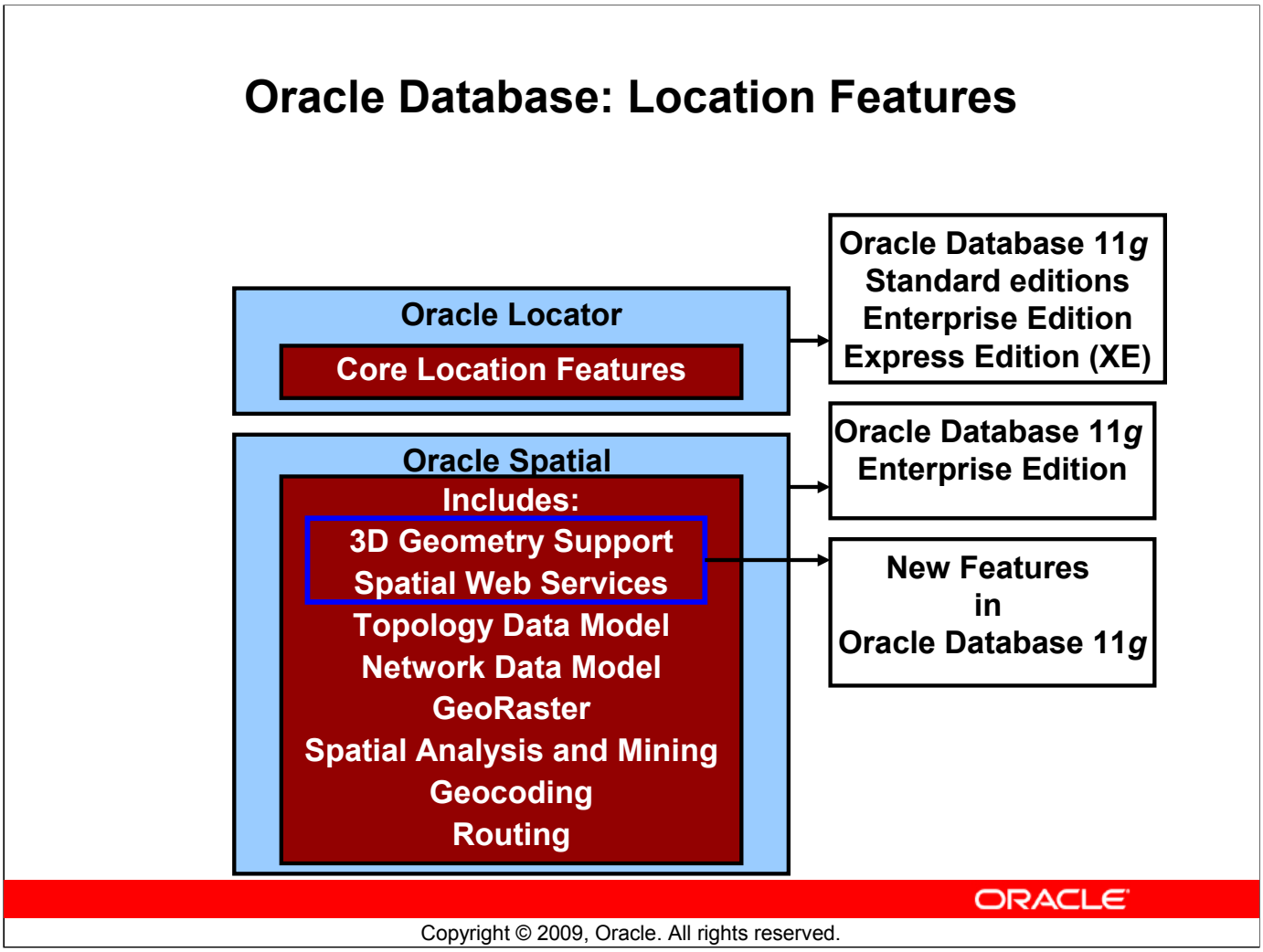
- Day 5:
  - Lesson 17: Managing Spatial Indexes
  - Lesson 18: Geocoding Address Data
  - Lesson 19: Using the Spatial Routing Engine
- Appendix A: Practice Solutions
- Appendix B: Overview of Oracle Workspace Manager

**This course can be taken by both 10*g* and 11*g* users.**

ORACLE

# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Spatial
- Overview of the object-relational data model
- Review of geometrical and geographical concepts
- The classroom environment and the example data set
- Oracle Spatial documentation and additional resources




## Oracle Database: Location Features

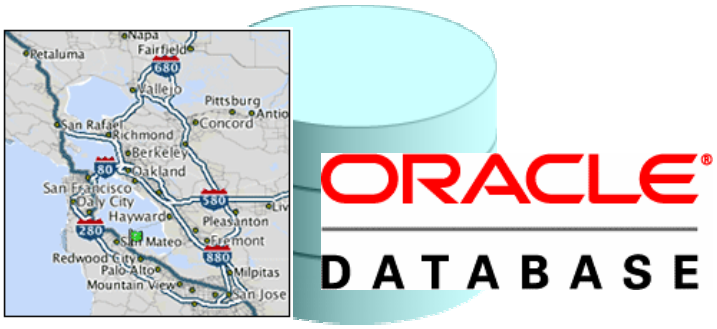
The location features in Oracle Database are provided by:

- **Oracle Locator:** Is a feature of Oracle Database 11*g* Standard, Enterprise, and Express (XE) editions that provides the core location functionality required by enterprisewide, location-based systems. Oracle Locator basically provides the core features and services that are available in Oracle Spatial. Core features include the data types, operators, and indexing capabilities of Spatial, along with a limited set of spatial functions and procedures.
- **Oracle Spatial:** Is a licensed option available with Oracle Database 11*g* Enterprise Edition that provides complete functional spatial features. If you want to design a high-end Geographic Information System (GIS), which requires complex spatial data management, you need Oracle Spatial. With Oracle Spatial, you have advanced capabilities such as linear referencing, spatial functions for spatial analysis and mining, topology data modeling, network data modeling, geocoding address data, routing, and support for GeoRaster data. Oracle Spatial also provides support for 3D geometries and spatial Web services.

**Note:** The support for 3D geometries and the Spatial Web Services technology are new in Oracle Spatial 11*g*. For more information about what features are available in Oracle Locator versus Oracle Spatial, refer to the Oracle Locator appendix in the *Oracle Spatial Developer's Guide.*

# Oracle Spatial: Spatial Data Management for Enterprise Applications

- Geospatial applications: High-end GIS
- Location-based services: Wireless location services or location-enabled business intelligence systems
- Interactive Web mapping: Public access to huge amounts of geographic data with an emphasis on aerial photography




ORACLE



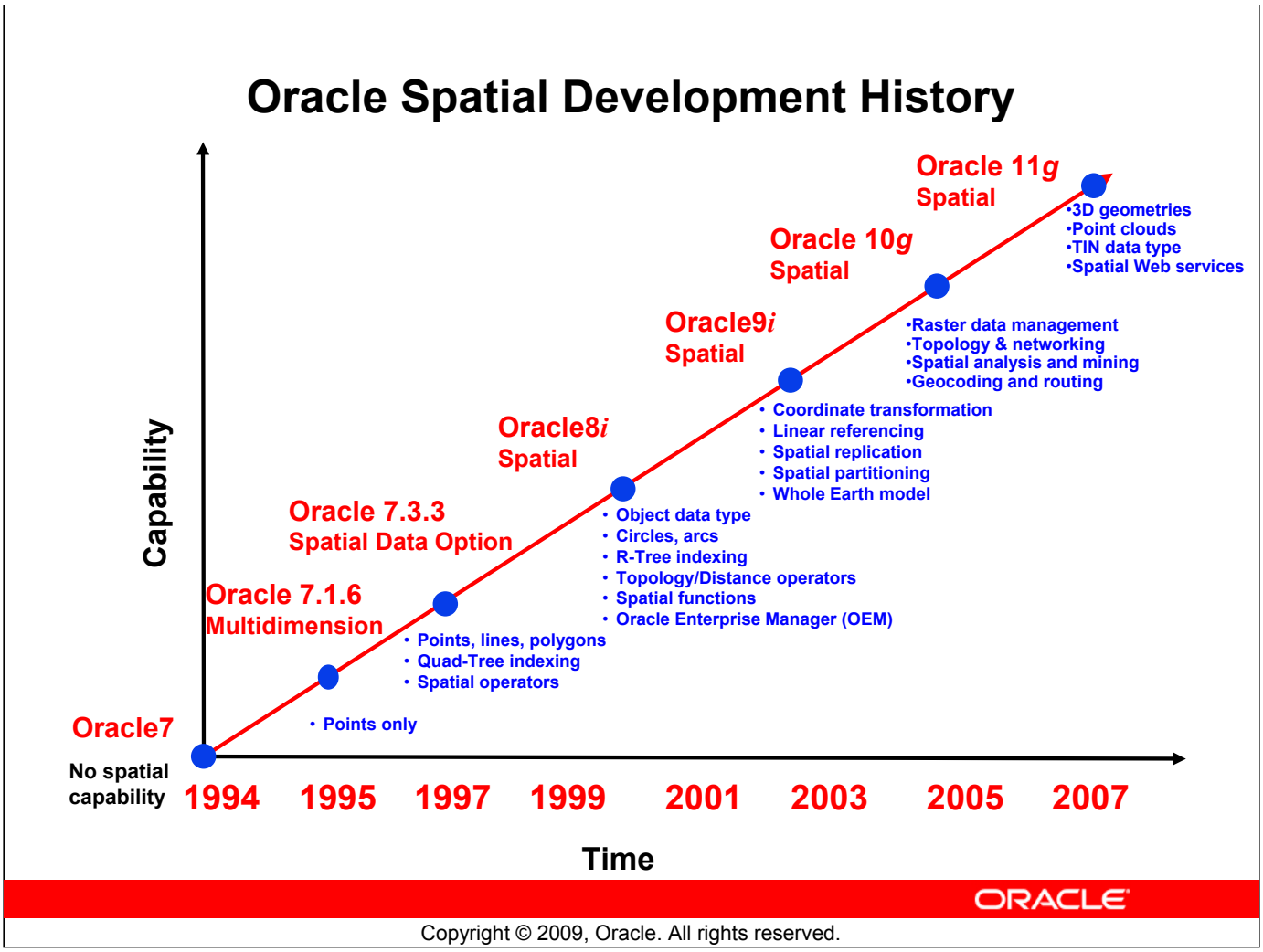
## Oracle Spatial: Spatial Data Management for Enterprise Applications

This slide is to give you an idea of the plethora of applications that use or are based on spatial data. Earlier maps, routes, or any kind of location-based data was available in some kind of atlas books or paper-based media. Today, on the Web, you can get an aerial view of your home through an online map. You have GIS-enabled cars that can drive you to your destination using the shortest route possible.

A GIS application is used for integrating, storing, editing, analyzing, sharing, and displaying geographically referenced information. The Geographic Information System technology can be used for scientific investigations, resource management, asset management, environmental impact assessment, urban planning, cartography, criminology, archaeology, sales, and marketing. For example, a business user may query a GIS application to decide on a new business to take advantage of an underserved market.

Location-based services are used to provide information that is specific to a location. These services are generally developed and provided by wireless carriers to enable mobile devices to display their location in relation to fixed assets (nearest restaurant, gas station, or fire hydrant) or mobile assets (friends, children, or police car), or to relay their position back to a central server for display or other processing purposes.

There are many mapping applications on the Web that publish huge amounts of geographic data with an emphasis on aerial photography. Oracle Spatial can also be used to store data from computer-aided design (CAD) and computer-aided manufacturing (CAM) systems.





## Oracle Spatial Development History

The product name has evolved over time: Multidimension (MD), Spatial Data Option (SDO), Spatial Cartridge, and finally Oracle Spatial.

Oracle Spatial started as a research and development effort with the Canadian Hydrographic Service (CHS) and Oracle Consulting. CHS sampled the ocean depth around Canada, and wanted to store and index many point data more efficiently in the Oracle database. This project evolved into the first version of the product called Multidimension (MD).

Oracle quickly realized that a point-only solution did not satisfy the requirements of most of its customers. In Oracle 7.3.3, the first version of the product that could store points, linestrings, polygons, polygons with holes, multipolygons, and multilinestrings was introduced. This was done in a purely relational data model.

- **Oracle7.3.3:** The product was called Spatial Data Option (SDO).
- **Oracle8:** The product was renamed Spatial Cartridge. Spatial Cartridge provided the same functionality as SDO (same relational data model, same API, minor improvements).
- **Oracle8*i*:** The name was changed to Oracle Spatial and it offered new features, including an object-relational data model for Oracle Spatial. Geometries in Oracle 8*i* are stored in a single row of a single column.
- **Oracle9*i*:** The product name was still Oracle Spatial and had new enhancements such as coordinate transformation, linear referencing, spatial replication, and partitioning.

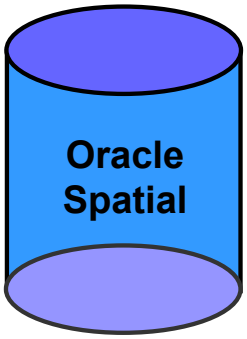
## Oracle Spatial Development History (continued)

- **Oracle 10*g*:** With Oracle 10*g*, Oracle Locator was defined as a subset of Oracle Spatial and was included in Oracle XE, Standard editions (including Standard Edition One) and Oracle Enterprise Edition. Oracle Spatial is a licensed option of the Enterprise Edition. Oracle Locator included core location features, whereas Oracle Spatial included advanced features such as geocoding, routing, topological data modeling, network data modeling, GeoRaster, and advanced spatial analysis and mining.
- **Oracle 11*g*:** The product name remains Oracle Spatial. Oracle Spatial 11*g* is a step forward in all the advanced areas that Oracle Spatial 10*g* introduced or supported. It is a complete spatial data management platform for geospatial or locations-based applications. Oracle Spatial 11*g* provides support for 3D geometries, point clouds, the triangulated irregular networks (TIN) data type, and includes enhanced support for spatial Web services.

## Oracle Spatial and Locator: Part of the Oracle DBMS Kernel

From the earliest Multidimension implementation through to today's Oracle Locator and Spatial products, Oracle Spatial technology:

- Is linked to the Oracle kernel
- Accesses and manipulates Oracle internal data structures
- Is written in C




ORACLE



### Oracle Spatial and Locator: Part of the Oracle DBMS Kernel

From the earliest Multidimension implementation through to today's Oracle Locator and Spatial products, Oracle Spatial technology:

- Is implemented directly in the Oracle kernel and is linked to it
- Accesses and manipulates Oracle internal data structures
- Is written in C, the same language that the Oracle database is written in

# Oracle Spatial Object-Relational Model

- Supports the object-relational model for representing geometries
- Has the following benefits:
  - A single object data type can represent many geometry types, including arcs, circles, compound polygons, compound line strings, and optimized rectangles.
  - It is easy to create and maintain spatial indexes and perform spatial queries.
  - Geometries can be stored in a single column of a normal Oracle table.
  - The performance is optimal.
  - It is easy to model geometrical data and queries.

ORACLE



## Oracle Spatial Object-Relational Model

The earlier Spatial option used relational data model to store and access geometries. Now, Spatial supports the object-relational model for representing geometries. With Oracle's extensible indexing, the object-relational model automates much of the work (for example, the index is maintained, queries are much simpler, and so on).

With the Oracle Spatial 11*g* release, the object-relational model has been enhanced to support three-dimensional (3D) geometries. New data types have been added to support surface and solids modeling.

# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Spatial
- Overview of the object-relational data model
- Review of geometrical and geographical concepts
- The classroom environment and the example data set
- Oracle Spatial documentation and additional resources

# Review: Oracle's Object-Relational Model

- Use abstract data types to group related columns and methods into objects.

```
CREATE OR REPLACE TYPE address_type as object
(street varchar2(50),
 zipcode number,
member function get_state()
return varchar2(25));
```

- Oracle creates a constructor method when the abstract data type is created.
- Use the `CREATE TYPE BODY` command to define the member functions.
  - For example, the `get_state` function is declared to return the State name.
  - The actual code for the function can be defined separately.

ORACLE



## Review: Oracle's Object-Relational Model

The following slides are included to provide a high-level overview of how the object-relational model is implemented in Oracle Database. It is helpful to know these concepts before you learn about Oracle Spatial's object-relational model in more detail.

Normally, in Oracle Spatial, you use predefined object data types and their methods. The Oracle Spatial object-relational model stores an entire geometry in the Oracle native spatial data type for the vector data named `SDO_GEOMETRY`. Primarily, you work with this native data type in this course, which is discussed in detail in the lesson titled "Overview of Oracle Spatial Concepts."

At times, you may need to create new object data types with embedded native spatial data types. You can use the example in the slide as a reference.

A constructor method instantiates an object of a particular object type. Oracle automatically creates a constructor method when the abstract data type is created. You may have multiple member functions for an object data type.

You can define the body of the member functions by using the `CREATE TYPE BODY` command.

## Review: Oracle's Object-Relational Model

- Define columns of a table of an object data type:

```
CREATE TABLE customer
(cust_id number,
 cust_addr address_type, ..);
```

- Call the default constructor while you insert values for the column of an object type:

```
INSERT INTO customer values
(100,
 address_type('Street_name', 11111), ..);
```

ORACLE



### Review: Oracle's Object-Relational Model (continued)

An Oracle table can contain columns of abstract data types. In Oracle Spatial, as you come across predefined native object data types, you create tables having columns based on these data types. For example, an Oracle table can contain one or more `SDO_GEOMETRY` columns. You also insert values for such columns in the table.

The slide examples show how to use abstract data types in an Oracle table.

## Review: Oracle's Object-Relational Model

- When you query the attributes of an object data type, fully identify the data type attributes by prefixing the column name with a table alias name.

```
SELECT cust_id, c.cust_addr.street
FROM customer c;
```

- Invoke the member function by using the column name that uses the object data type.

```
SELECT c.cust_addr.get_state()
FROM customer c;
```

ORACLE



### Review: Oracle's Object-Relational Model (continued)

The slide examples explain how to access abstract data–type attributes in a query and how to invoke a member method of an abstract data type. Usually, this is how you write spatial queries that involve native spatial data types and their methods.

Here, the `get_state` function is invoked using the column name.

# Review: Using Varying Arrays

- A varying array, or a varray, enables you to store repeating attributes of a record in a single row—for example, a customer may have up to two addresses.
- Create a varray based on either an abstract data type or one of Oracle's standard data types.

```
CREATE OR REPLACE TYPE address_va
AS varray(10) of address_type;
```

- Define varrays with the maximum capacity that may be required to hold repeating attributes of a record.

ORACLE



## Review: Using Varying Arrays

In Oracle Spatial, you come across a few predefined varying arrays, commonly referred to as varrays. This slide provides you an overview of varrays. If you want to create a varray based on the `address_type` data type, execute the following command:

```
CREATE OR REPLACE TYPE address_va AS varray(10) of address_type;
```

This enables you to store multiple addresses for each customer. Varrays must have the maximum capacity that may be required to hold repeating attributes of a record. For example, a customer may have a maximum of 10 addresses. If you store only two addresses for a customer, the other eight are uninitialized.

# Review: Using Varying Arrays

- Use varray as a type for a column in a table:

```
CREATE TABLE customer_new(
Customer_id number,
Cust_addr address_va);
```

- You must nest calls to multiple constructor methods to insert a record into a table.

```
INSERT INTO customer_new
values (102, address_va(address_type
                           ('Street_name1',12345),
                        address_type
                           ('Street_name2',56789)));
```

ORACLE



## Review: Using Varying Arrays (continued)

The customer table's address column must be of the `address_va` type.

To insert new values, you must call the default constructor of the varray, which in turn must call the default constructor of the abstract data type that the varray is based on. The slide example shows how you insert multiple address values for each customer.

You can easily query varrays in a table, as follows:

```
SELECT cust_addr FROM customer;
```

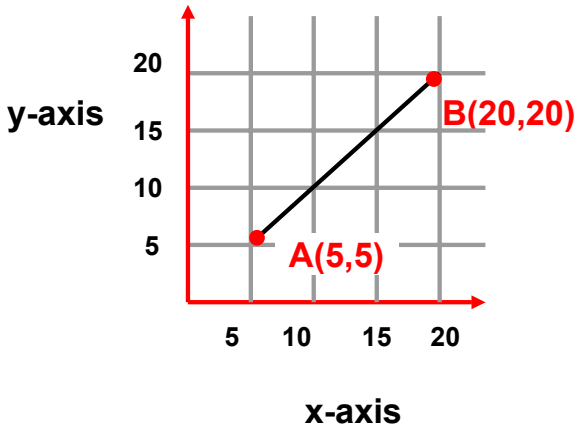
It lists all address values for all customers.

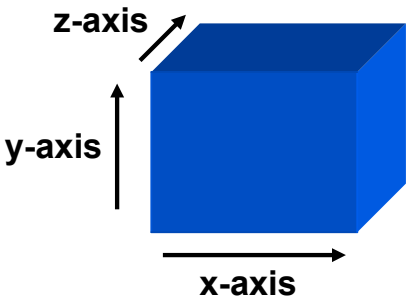
# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Spatial
- Overview of the object-relational data model
- Review of geometrical and geographical concepts
- The classroom environment and the example data set
- Oracle Spatial documentation and additional resources

# Geometries Along Axes

- Two-dimensional (2D) geometries are represented by pairs of x-axis and y-axis ordinates.
- Three-dimensional (3D) geometries are represented by a set of three values: x-axis, y-axis, and z-axis ordinates.



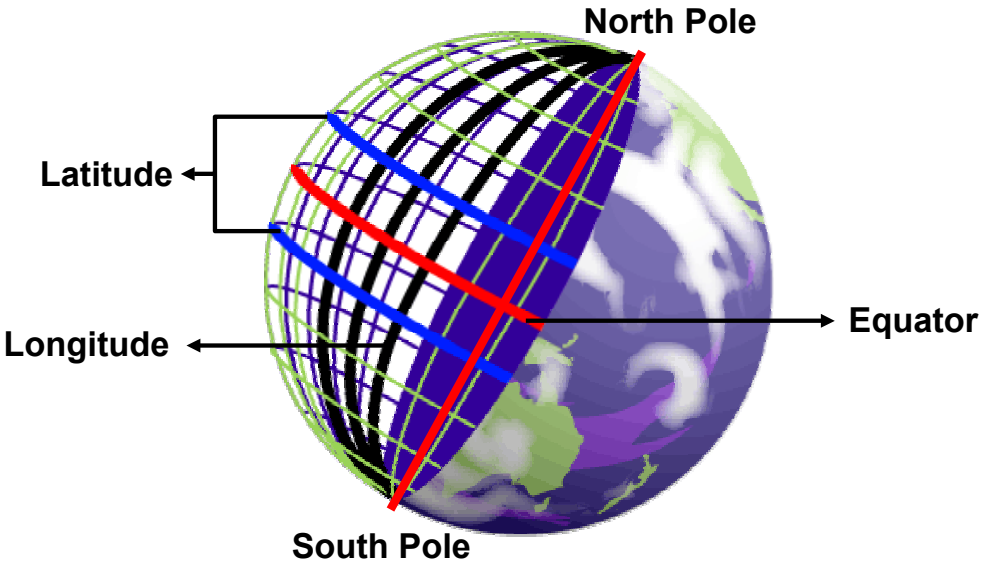

ORACLE



## Geometries Along Axes

Geometry is an ordered sequence of vertices that are connected by straight-line segments or circular arcs. Two-dimensional points are composed of two ordinates: X and Y, whereas three-dimensional points are composed of three ordinates: X, Y, and Z.

# Common Geographical Terms Used in the Course




ORACLE



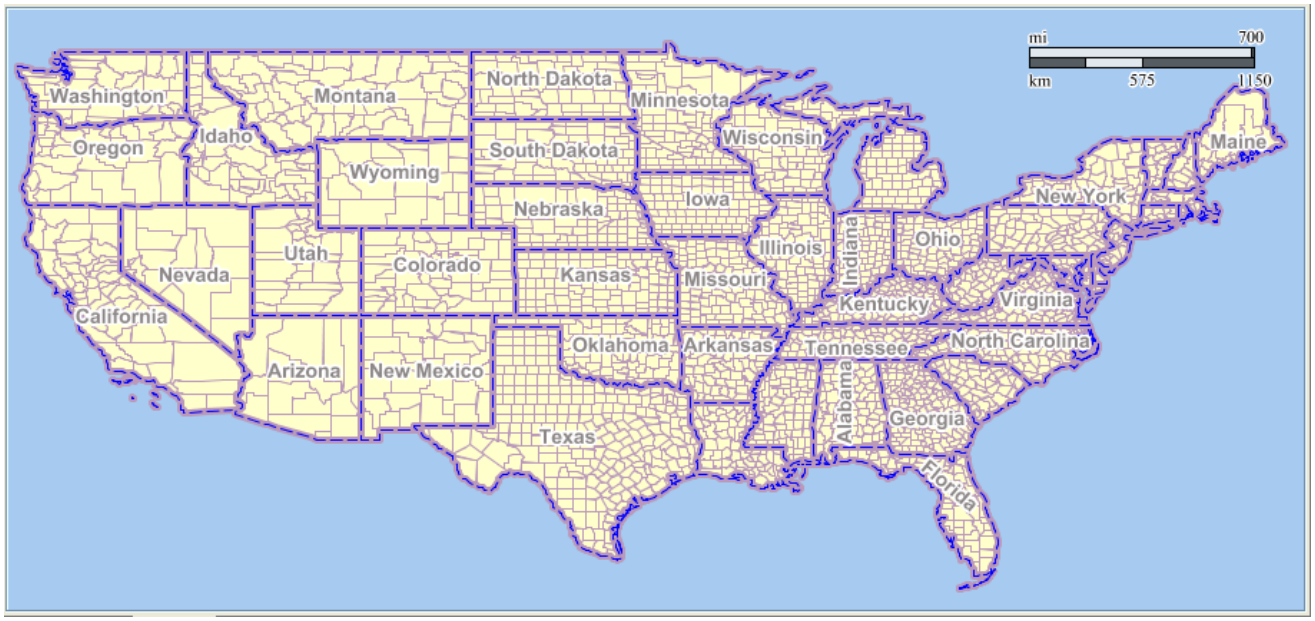
## Common Geographical Terms Used in the Course

Here are some of the common geographical terms used in this course. Use this slide to review some of the concepts or terms that you may already know. The Oracle Spatial concepts rely on these terms and it is good for you to visually and conceptually refresh your knowledge of geographical terms.

Some of the definitions of common terms are as follows:

- **Equator** is an imaginary circle around the earth, halfway between the North Pole and South Pole.
- **Latitude** is the angular distance north or south from the equator to a particular location. The equator has a latitude of zero degrees, whereas the North Pole has a latitude of 90 degrees north, and the South Pole has a latitude of 90 degrees south.
- **Longitude:** Longitude is the angular distance east or west from the north-south line that passes through Greenwich, England, to a particular location. Greenwich, England has a longitude of zero degrees. The farther east or west of Greenwich you are, the greater is your longitude.

# Political Divisions of the United States




ORACLE



## Political Divisions of the United States

The example data set used in this course is based on the political divisions of the United States. If you belong to any other country, these units and divisions may not come naturally to you. This slide is to give you an understanding of what each political division means. It helps you visualize the results of spatial queries when you run them later in the course.

The United States (US) is divided into 50 states. The states include California, Texas, Florida, and so on.

Each of the fifty states of the United States of America is typically divided into counties. A county of the United States is a local level of government smaller than a state, but not smaller than a city or town. That is, multiple cities or towns are governed on a local level as counties. A county may be referred to as a "parish," a "borough," or a "district" in other parts of the world.

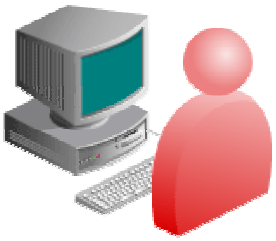
For example, the state of Delaware is subdivided into three counties: (from north to south) New Castle, Kent County, and Sussex. Each county comprises many cities and towns. For example, New Castle County comprises about 15 cities or towns, such as Townsend, Wilmington, Delaware city, and so on.

# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Spatial
- Overview of the object-relational data model
- Review of geometrical and geographical concepts
- The classroom environment and the example data set
- Oracle Spatial documentation and additional resources

# Classroom Environment

- Windows XP platform
- Oracle Database 11*g* Enterprise Edition (The Spatial and Partitioning options are installed by default.)
- Database instance: `orcl`
- User account name: `student`
- Oracle Database 11*g* Examples media
  - Installed for Spatial demos
- Additional software downloaded:
  - MapViewer Quickstart kit
  - MapViewer Demo Data set
  - Map Builder kit



ORACLE

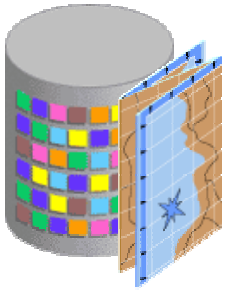


**Classroom Environment**

This slide describes the classroom setup.

# Example Data Set Used in This Course

- Use the MapViewer Demo Data Set.
- For the Geocoding and Routing lessons, use Washington DC the sample data sets provided by ADCI.
  - Sample data available at http://www.adci.com/oracle/
- Use Oracle Spatial demos.
  - Shipped with Oracle Database 11*g* Examples media



ORACLE



## Example Data Set Used in This Course

American Digital Cartography Inc. (ADCI) provides NAVTEQ data for users of Oracle Spatial or Locator, or both. Sample data set used in this course is available online for testing purposes at the following URL:

http://www.adci.com/oracle/

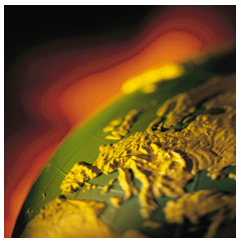
Apart from the preceding data sets, this course also uses sample data sets provided in the form of Data Pump (DMP) files and SQL*Loader files. These files will be provided to you in the classroom setup.

# Lesson Agenda

- Course objectives, agenda, and appendixes used in the course
- Overview of Oracle Spatial
- Overview of the object-relational data model
- Review of geometrical and geographical concepts
- The classroom environment and the example data set
- Oracle Spatial documentation and additional resources

## Oracle Spatial Documentation and Resources

- *Oracle Spatial Developer's Guide*
- Oracle Spatial on OTN*:*
  - http://www.oracle.com/technology/products/spatial/
- MetaLink (http://metalink.oracle.com) for:
  - Spatial patches
  - Useful error and workaround information
- Oracle Application Server MapViewer on OTN:
  - http://www.oracle.com/technology/products/mapviewer/



ORACLE



### Oracle Spatial Documentation and Resources

- Navigate to the online Oracle Database 11*g* Documentation Library using the following URL:

  http://www.oracle.com/pls/db111/homepage
- Expand the "Unstructured Data and Content Management" navigation folder and click the Oracle Spatial and Location Information link to access the documentation.
- You can post your queries and look for common errors and their solutions on the discussion forums accessible from the Oracle Spatial page on OTN.
- **MetaLink (http://metalink.oracle.com):** Visit MetaLink (http://metalink.oracle.com) to download Oracle Spatial patches. Oracle Spatial is part of the core patch set and is available on all platforms supported by the core patch set. To download patches from MetaLink, click the Patches button.

# Spatial Patches

- Oracle Spatial patches are moving to a new model, patch bundles, beginning with Oracle 10.2.0.4.
- The patch bundle numbers will not change, even if they are updated to include more spatial patches.
- If you have previously downloaded a spatial patch bundle, and the bundle is updated to include another spatial patch, you will be notified by email.
- Occasionally, you may need a spatial patch that is not yet part of the bundle.
- Both, spatial patch bundles and individual patches are made available on MetaLink.

ORACLE



**Spatial Patches**

The slide gives you an overview on the way Spatial patches work. It explains how you can use the new Spatial patch bundles.

# Spatial Patches

The following patches and patch bundles are recommended for bug fixes and better performance, and are available on MetaLink.

- For Oracle 10.2.0.3, patches:
  - 6980648 – Bug fixes and improved performance
  - 6329260 – Bug fixes and improved performance
  - 5888136 – Only if you are running Solaris; improved performance
  - 6956194 – Bug fix if working with geodetic data
  - 6868237 – To add a `WKTEXT` coordinate system with `CS_SRS`
- For Oracle 10.2.0.4, patch bundle
  - 7641182 – If not available on your hardware platform, open a service request to request a back port.

ORACLE



## Spatial Patches (continued)

The patch bundle number for 10.2.0.4 is listed in this slide. Some of the most important patches that are available on MetaLink, and also included in the patch bundles, are listed as follows. Some of the patches listed here conflict with each other. However, conflicts are resolved in patch bundles. If a bundle is not yet available on MetaLink for your hardware platform, but an individual patch is, apply the individual patch. Also log a service request for the patch bundle on your hardware platform.

For Oracle 10.2.0.3, patches:

- 6980648 – Bug fixes and improved performance
- 6329260 – Bug fixes and improved performance
- 5888136 – Only if you are running Solaris; improved performance
- 6956194 – Bug fix if working with geodetic data
- 6868237 – To add a `WKTEXT` coordinate system with `CS_SRS`

For Oracle 10.2.0.4, patches:

- 7003151 – Bug fixes and improved performance
- 6989483 – Improved performance
- 7046751 – Apply this patch if you applied patch 6989483
- 7237687 – Bug fix
- 7276032 – Bug fix
- 6868237 – To add a `WKTEXT` coordinate system with `CS_SRS`
- 6501889 and 6504890 – Apply these patches if working with local partitioned spatial indexes

# Spatial Patches

- For Oracle 11.1.0.6, patch bundle:
  - 7656107 – If not available on your hardware platform, open a service request to request a back port.
- For Oracle 11.1.0.7, patch bundle:
  - 7656113 – If not available on your hardware platform, open a service request to request a back port.
- For MapViewer 10.1.3.1:
  - 7384506 – Bug fixes and improved performance

ORACLE



## Spatial Patches (continued)

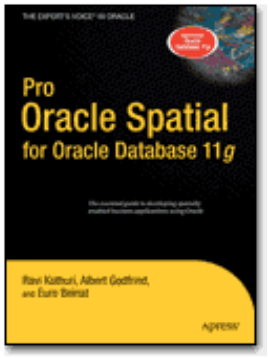
The patch bundle numbers for 11.1.0.6 and 11.1.0.7 are listed in this slide.

For Oracle 11.1.0.6, patches:

- 7003151 – Bug fixes and improved performance
- 6989483 – Improved performance
- 7046751 – Apply this patch if you applied patch 6989483
- 6956194 – If working with geodetic data
- 6868237 – To add a WKTEXT coordinate system with CS_SRS
- 6501889 and 6504890 – Apply these patches if working with local partitioned spatial indexes

# Additional Resources

- *Pro Oracle Spatial for Oracle Database 11g*
  - Ravikanth V. Kothuri, Albert Godfrind, and Euro Beinat




- Oracle Spatial Demos
  - Install Oracle Database 11*g* Examples media (formerly, Companion CD)

ORACLE



## Additional Resources

Examples and demos of Spatial are available on the Oracle Database Examples media.

For practical examples of developing applications by using Oracle Spatial and MapViewer, refer to *Pro Oracle Spatial for Oracle Database 11g*.

# Summary

Oracle Database location features are:

- Oracle Locator: Consists of core location features
- Oracle Spatial: Is the superset of Oracle Locator with advanced spatial capabilities such as support for 3D geometries, spatial Web services, topological and network modeling, geocoding, routing, and support for the GeoRaster technology

ORACLE



## Summary

In this lesson, you learned about the history and package options in the Oracle Spatial or Locator technology. This lesson briefly reviewed the basic concepts and terminologies of the object-relational data model. The lesson also covered the organization of the course and the classroom environment used in this course.

# 1

# Overview of Oracle Spatial Concepts

ORACLE

# Objectives

After completing this lesson, you should be able to:

- Describe the supported geometric primitive types
- Define the Oracle Spatial data model
- Describe the Spatial query model
- Describe the concept of Spatial indexing
- Describe coordinate systems
- Define Spatial queries
- Describe Linear Referencing System (LRS)
- Explain geocoding and routing
- Define Oracle Application Server MapViewer

ORACLE



## Objectives

In this lesson, you get a broad overview of the concepts and terminologies associated with the Oracle Spatial technology. You get a basic understanding of the Oracle Spatial concepts.

You are not expected to fully understand all the concepts discussed in this lesson. All concepts are discussed in more detail in later lessons.

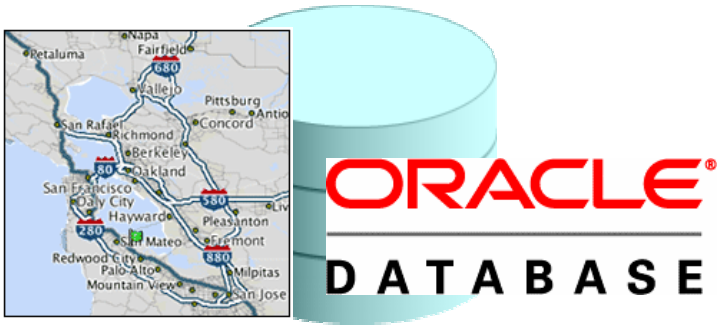
# Lesson Agenda

- Oracle Spatial and the supported geometric primitive types
- Spatial data model: Elements, geometries, and layers
- Define:
  - Coordinate systems
  - Spatial query
  - Spatial indexing
  - Tolerance
- Spatial query model: Primary and secondary filters
- Introduction:
  - Linear Referencing System
  - Geocoding and Routing Engine
  - Oracle Application Server MapViewer

ORACLE

# Oracle Spatial

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




ORACLE



## Oracle Spatial

Oracle Spatial is defined in this slide. Spatial data is any data that represents the essential characteristics of the location of real or conceptual objects as those objects in relation to the real or conceptual space in which they exist.

As a core location feature of Oracle Database, Oracle Spatial or Oracle Locator essentially provides a schema (`MDSYS`) and a set of functions and procedures that facilitate the storage, retrieval, update, and query of collections of spatial features or data in an Oracle database.

As explained in the introductory lesson, Oracle Locator is a subset of Oracle Spatial because it provides the core location features that are available in Oracle Database.

# Examples of Spatial Data

- Road map: Is a two-dimensional object that contains points, lines, and polygons to represent cities, roads, and state boundaries respectively
- Earth-relative spatial data: Includes longitude and latitude values that indicate a location on Earth. For example:
  - Customer location
  - Store location
  - Hospital location
- CAD/CAM systems data: Operates on a different scale and precision of data. For example:
  - Floor plans
  - Parts of an automobile or an airplane

ORACLE



## Examples of Spatial Data

Spatial data is mostly used by location-enabled and Geographic Information System (GIS) applications. A common example of spatial data is a road map that contains two-dimensional points, lines, and polygons to represent cities, roads, and state boundaries. The location of cities, roads, and political boundaries are projected onto a two-dimensional display or piece of paper, preserving the relative positions and relative distances of the rendered objects.

GIS applications store, retrieve, and render Earth-relative spatial data in terms of longitude and latitude.

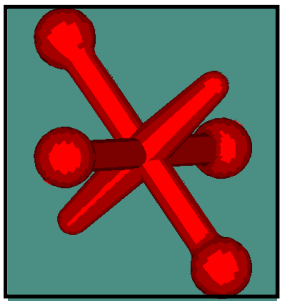
The types of spatial data that can be stored using Oracle Spatial include location-based and GIS data as well as data from computer-aided design (CAD) and computer-aided manufacturing (CAM) systems.

CAD and CAM systems do not work on a geographic scale; rather, they work on a smaller scale, such as for an automobile engine or printed circuit boards. They operate on a different scale and precision of data and may be more complex at times.

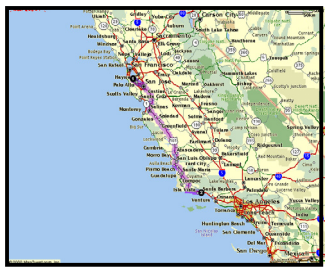
# Types of Location Data

GIS (mapping) data

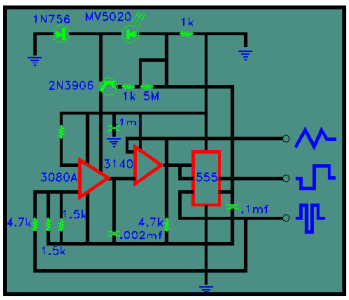
CAD data



Points of interest



CAM data




ORACLE

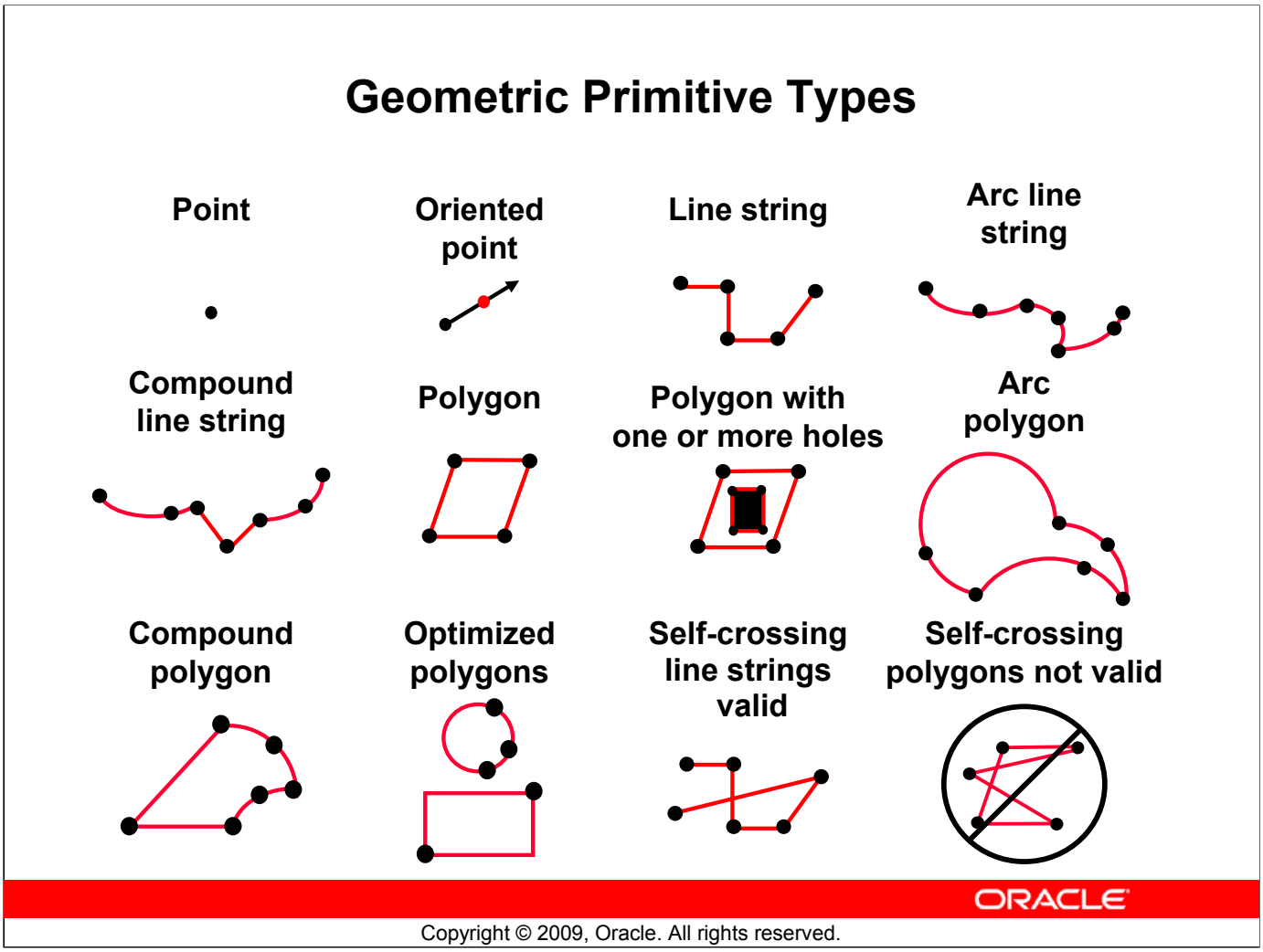


## Types of Location Data

Oracle Spatial is designed to make the storage, retrieval, and manipulation of spatial data easier and more natural to users. After this data is stored in an Oracle database, it is easily and meaningfully manipulated and retrieved in the same manner as any other data stored in the database.

Spatial information is displayed in maps and has two components: location and attributes. Location is defined by coordinates that define the position of a feature. The use of longitude and latitude, such as W77.29 and N39.02, is an example of specifying geographic information using a global coordinate system. Attributes represent the characteristics of the feature. Examples of attributes of a feature include the owner, identification number, tax assessment, the date on which the data was collected, or the land use code of an area.

The difference is only in the scale of the data and not in its complexity. It may 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. However, if the diameter of an engine's pistons is off by a few tenths of an inch, the engine does 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.




## Geometric Primitive Types

Oracle Spatial supports many geometric primitive types. A point may represent a building location, a line string may represent a road or flight path, and a polygon may represent a state, city, zoning district, or city block.

Oracle Spatial supports two-dimensional geometric types that are composed of one or more pairs of X and Y ordinates. Each of these types is discussed in detail in the following slides.

**Note:** With Oracle Database 11*g*, Spatial also supports the storage and indexing of three-dimensional geometric types, where three coordinates are used to define each vertex of the object being defined.

# Points and Oriented Points

- Points (X1, Y1):
  - Define a location in X and Y
  - Are used to represent point features
  - Are often used to track moving objects
  - Can be used to represent locations of buildings, customers, ATMs, or automobiles
- Oriented points (X1, Y1, X2,Y2):
  - Define a point location by X1, Y1 coordinates
  - Define an orientation vector by X2, Y2 coordinates
  - Are typically used for orienting symbology for maps and other display applications


ORACLE



## Points and Oriented Points

### Points

- A coordinate can be 2D, 3D, or 4D. Points are elements that are composed of between two and four ordinates. Often, if there are only X and Y coordinates, they correspond to either longitude and latitude or X and Y on a Cartesian plane.
- A point could represent a building, a customer location, or an ATM.

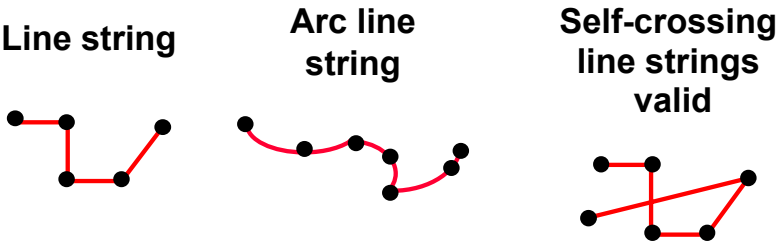
### Oriented Points

- Include a point coordinate that can be 2D, 3D, or 4D, and represent a location
- Also include an orientation vector, which typically represents how to orient symbology in a display application

**Note:** Oriented point support is described in more detail in the lesson titled "Defining Collection Geometries."

# Line Strings

- Line strings (X1, Y1, …, X*n*, Y*n*):
  - Are represented by a series of connected points
  - Are connected by either straight lines or circular arcs
- The boundary of a line string is its end points.
- Lines that close to form a ring have no area.
- Self-crossing lines are supported.
- Line strings can be used to represent rivers, roads, cables, pipelines, and other linear features.




ORACLE

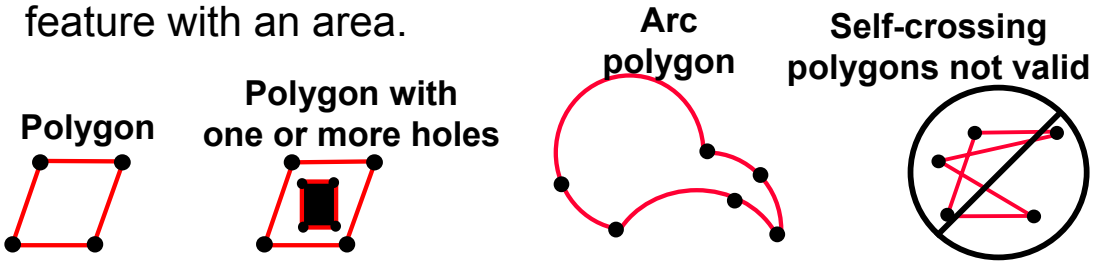


## Line Strings

- A line string is a series of two or more points that are connected by either straight lines or circular arcs. When connected by arcs, it is termed arc line string.
- Line strings can cross themselves.
- Line strings have no area, even if they close to form a ring.
- The boundary of a line string is its end points. If the end points match (that is, the line string closes on itself), it has no boundary.
- A line string could represent a river, a road, a utility cable, or any other linear feature.

# Polygons

- Polygons (X1, Y1, …, X*n*, Y*n*):
  - Are area features formed by a series of connected points
  - Can be connected by all straight lines or all circular arcs
- The first point is the same as the last point.
- The interior area can include a void defined by an additional polygon (for example, a polygon with a hole).
- Self-crossing polygons are not supported.
- A polygon can represent a land parcel, region, park, or any feature with an area.






## Polygons

- A polygon is an area feature, represented by a series of connected points, where the first point is the same as the last point (repeating the last point is an Open Geospatial Consortium [OGC] standard requirement). Coordinates must close and an interior area is implied.
- Polygons can contain voids using additional polygons that represent holes.
- Polygons can be made up of all straight lines or all circular arcs.
- Self-crossing polygons are not supported although self-crossing line strings are. If a line string crosses itself, it does not become a polygon. If a polygon touches or crosses itself, it is invalid.
- This rule and many others regarding the validity of polygons are defined by the OGC.

# Polygons: Optimized Circles and Rectangles

- An optimized circle:
  - Is represented by three distinct points on the circumference of the circle
  - Has an area
- An optimized rectangle:
  - Is represented by the lower-left point and the upper-right point of the rectangle
  - Has an area


ORACLE

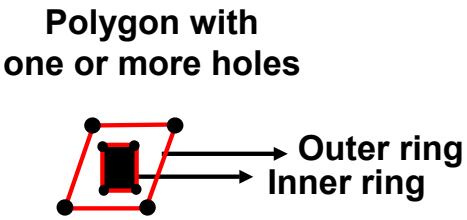


## Polygons: Optimized Circles and Rectangles

- Optimized circles and optimized rectangles are also categorized as polygons because they have an area associated with them.
- An optimized circle is represented by three distinct points on the circumference of the circle.
- An optimized rectangle is represented by the lower-left point and the upper-right point of the rectangle.

# Polygons: Outer Ring and Inner Ring

- An outer ring must be specified in counterclockwise rotation.
- An inner ring describes a void.
  - It is specified in clockwise rotation.
- An outer ring must be followed by all its inner rings.




ORACLE



## Polygons: Outer Ring and Inner Ring

- An outer ring must have a counterclockwise rotation.
- Voids (inner rings) are stored with a clockwise rotation
- An outer ring must be followed by all its inner rings.

Other rules about polygons include:

- If the polygon has one or more voids, all voids must be contained totally within the polygon.
- Voids cannot overlap.
- Voids may touch each other at one point. The boundary of the outer ring may be touched by a void at only one point.

# Polygons: Outer Ring and Inner Ring

- A spatial migration tool is available to correct the rotation and ordering of rings of a polygon.
- The migration tool:
  - Generates polygons that follow rules about rotation
  - Orders the outer ring immediately followed by all its inner rings

ORACLE



## Polygons: Outer Ring and Inner Ring (continued)

A spatial migration tool fixes polygons that may not have correct rotation and ring ordering. It generates polygons that follow rules about rotation and ring ordering.

## Compound Line Strings and Polygons

- Compound line strings (X1, Y1, …, X*n*, Y*n*):
  - Are a combination of straight lines and circular arcs
- Self-crossing compound line strings are supported.
- Compound line strings that close to form a ring have no area.
- Compound polygons (X1, Y1, …, X*n*, Y*n*):
  - Are a combination of straight lines and circular arcs
  - Have an area
  - Have the same semantics as polygons


ORACLE



### Compound Line Strings and Polygons

#### Compound Line Strings

- The line strings discussed earlier were either all connected with straight lines or all connected with circular arcs. A compound line string is a combination of straight lines and circular arcs. With a compound line string, you can define a single contiguous line string that is made up of straight lines and circular arcs.
- They have the same properties as line strings. Self-crossing lines are supported and no area is ever implied, even if it closes to form a ring.

#### Compound Polygons

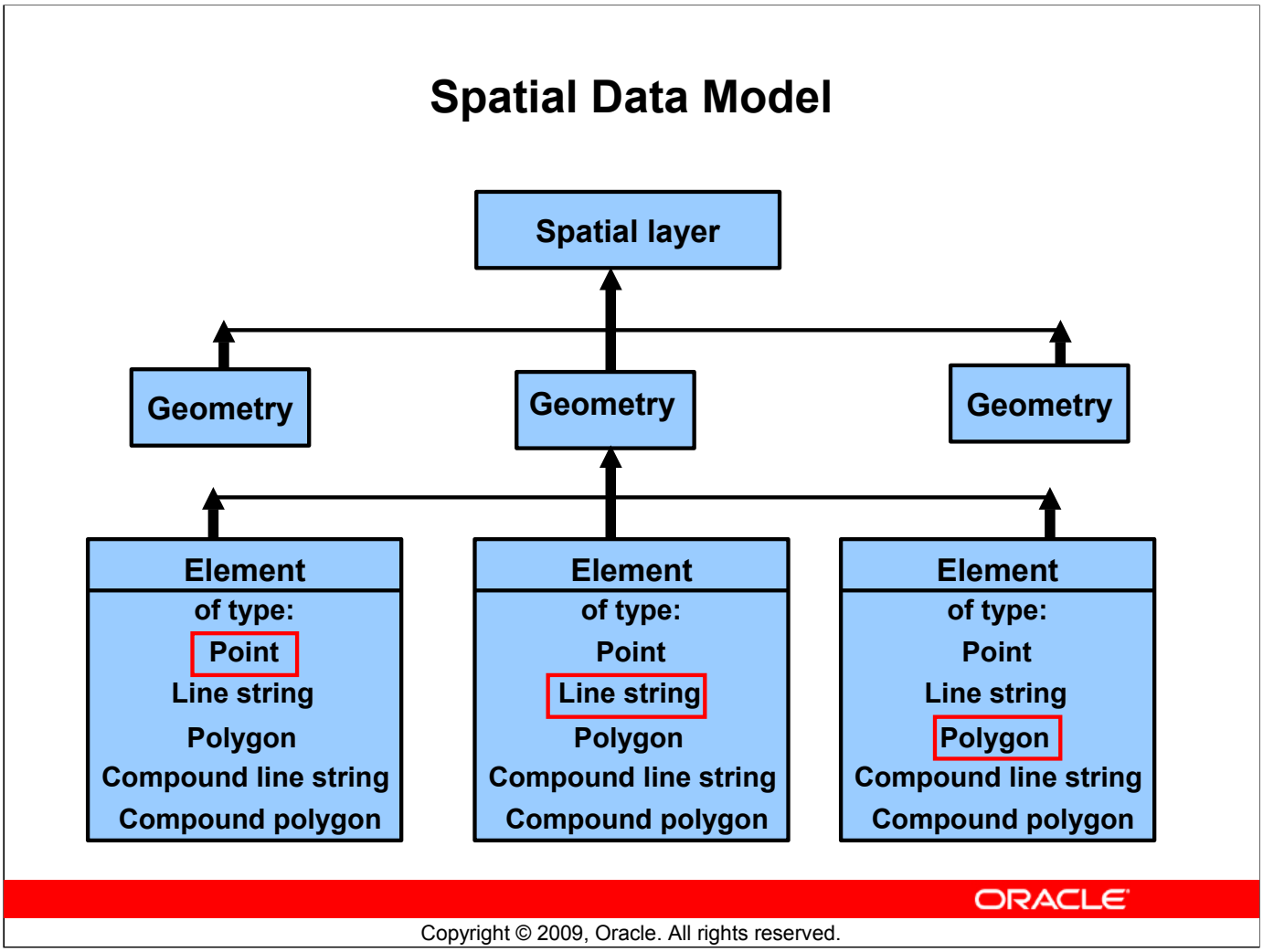
- The polygons discussed earlier were either all connected with straight lines or all connected with circular arcs. A compound polygon is a combination of straight lines and circular arcs. With a compound polygon, you can define a single contiguous polygon that is made up of straight lines and circular arcs.
- A compound polygon is often the result of a buffer operation. If you buffer a line, you are likely to get a compound polygon (that is, straight lines on either side of the line that you buffered and half circles around the end points).
- They have the same semantics as described for polygons (they cannot self-cross, the direction of rotation is important, and so on).

**Note:** Compound geometries were added to support the buffer operation.

# Lesson Agenda

- Oracle Spatial and the supported geometric primitive types
- Spatial data model: Elements, geometries, and layers
- Define:
  - Coordinate systems
  - Spatial query
  - Spatial indexing
  - Tolerance
- Spatial query model: Primary and secondary filters
- Introduction:
  - Linear Referencing System
  - Geocoding and Routing Engine
  - Oracle Application Server MapViewer

ORACLE




## Spatial Data Model

The Oracle Spatial data model is a hierarchical structure.

Working from the bottom to the top, an element is synonymous with a geographic primitive type, as discussed in previous slides. An element can be of one of the following types: point, line string, polygon, compound line string, or compound polygon.

A geometry is made up of one or more elements. A few examples of geometries that are made up of multiple elements are:

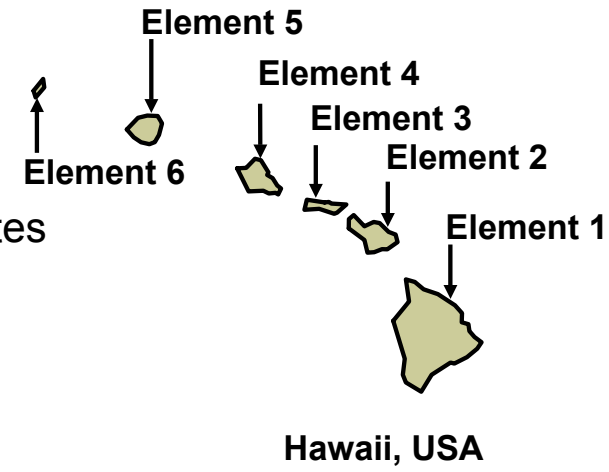
- **6th Avenue in New York City:** 6th Avenue starts in southern Manhattan, hits Central Park, and continues on the other side of Central Park. This is an example of a geometry with two noncontiguous line string elements.
- **U.S. State of Hawaii:** The state of Hawaii is composed of multiple islands. This is an example of a geometry with several polygon elements.

The elements that make up a geometry do not all have to be of the same element type. For example, a geometry can be composed of a point element and a polygon element.

A spatial layer is an `SDO_GEOMETRY` data type column in an Oracle table. A layer is made up of many geometries, each stored as a separate row. Data modeling is extremely important. When you define your table that contains a geometry column, you must ensure that all the other columns in that table have a one-to-one relationship with that geometry. Typical Oracle data modeling concepts apply. Sets of geometries are stored in a spatial layer because they share common attributes.

# Element

- Is the basic building block of a geometry
- Supports the following spatial element types:
  - Point
  - Line string
  - Polygon
  - Compound line string
  - Compound polygon
- Is constructed using ordinates






## Element

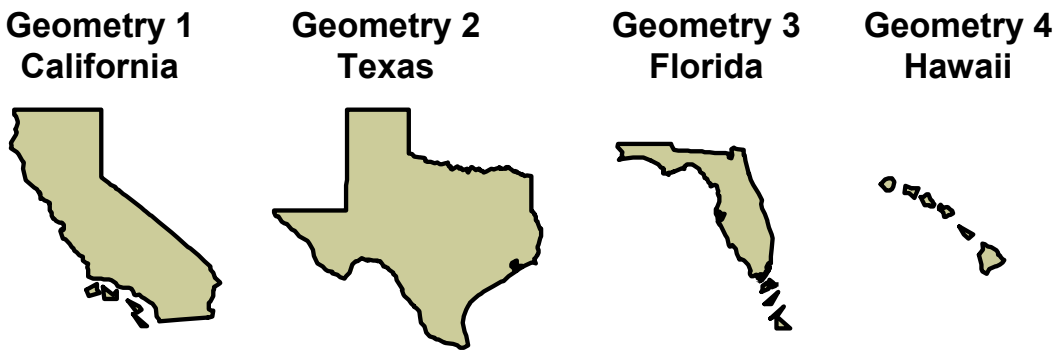
Elements are the building blocks of a geometry and are similar to the geometric primitive types discussed earlier. The supported spatial element types are: point, line string, polygon, compound line string, and compound polygon.

If you defined a point in a two-dimensional coordinate system, the point is referred to as a coordinate. X by itself is an ordinate and Y by itself is an ordinate. Elements are constructed using ordinates, and it is very common in Oracle Spatial to refer to the ordinates of a geometry.

The example in the slide (Hawaii, USA) is a geometry made up of six polygon elements.

# Geometry

- Represents a spatial feature such as states, roads, or a set of islands
- Consists of an ordered set of primitive elements
- Can be formed by using elements of a single primitive type or using elements of different primitive types




ORACLE



## Geometry

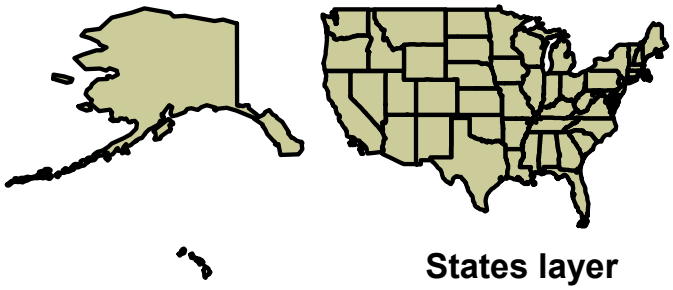
A geometry is the representation of a user's spatial feature, modeled as an ordered set of geometric primitive elements.

Each of the geometry examples in the slide is an example of a U.S. state (California, Texas, Florida, and Hawaii) that is made up of multiple elements, where all the elements are of the same type. A homogeneous geometry is formed of elements of a single primitive type. For example, a set of islands is formed of polygons. Geometries that contain different element types are also supported by Oracle Spatial. A heterogeneous geometry is one that is formed of different types of element, such as a point and a polygon.

A polygon with holes is stored as a sequence of polygon elements in a geometry. The exterior rings of the polygons cannot overlap.

# Spatial Layers

- Consist of geometries that share a common set of attributes
- Are geometry columns of the SDO_GEOMETRY native data type in a table
- Should have a one-to-one relationship with other columns in the table



States layer

ORACLE



## Spatial Layers

A layer is synonymous with a column in a table. Just as there are NUMBER, VARCHAR2, and DATE data types, there is also a native data type called SDO_GEOMETRY. An Oracle table can have multiple SDO_GEOMETRY columns (that is, multiple layers). For instance, a table may contain all the states in the United States at a low resolution as well as at a high resolution. The application can determine which layer to query depending on the zoom level. The States layer shown in the slide can be modeled in a table, which contains 50 rows, one for each U.S. state.

An SDO_GEOMETRY object (one column, one row) can contain one or more elements, where each element can be any of the geometric primitive types discussed in this lesson.

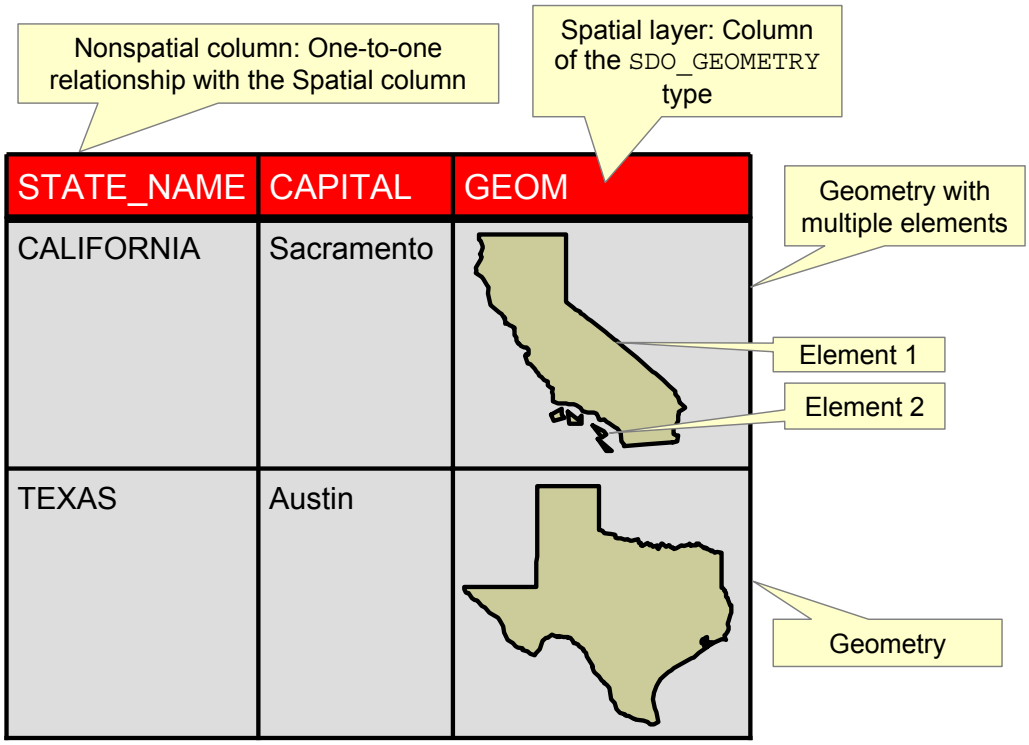
Data modeling is very important, but is no different from ordinary Oracle data modeling. You want to make sure that the SDO_GEOMETRY column in your table has a one-to-one relationship with the other columns in that table. This should not be a new concept for anyone who knows how to create a normalized table.

**Note:** You learn about the SDO_GEOMETRY native data type in the lesson titled "Creating Spatial Layers."

## Spatial Layers (continued)

For example, roads may have a property called lanes, which does not apply to a river because rivers do not have lanes, and rivers may have a property called salinity, which does not apply to a road because a road does not have salinity. Roads and rivers do not share a common set of attributes. It is more effective to model these as two layers in their own tables—one table for roads that contains a geometry column and additional attribute columns related to roads, and one table for rivers that contains a geometry column and additional attribute columns related to rivers.

# Spatial Data Model Represented in an Oracle Table




ORACLE



## Spatial Data Model Represented in an Oracle Table

A Spatial layer is the `SDO_GEOMETRY` column that stores geometries that share a common set of nonspatial attributes. A geometry is stored in a cell of the table. This geometry may be formed of an ordered set of primitive elements.

# Lesson Agenda

- Oracle Spatial and the supported geometric primitive types
- Spatial data model: Elements, geometries, and layers
- Define:
  - Coordinate systems
  - Spatial query window
  - Spatial indexing
  - Tolerance
- Spatial query model: Primary and secondary filters
- Introduction:
  - Linear Referencing System
  - Geocoding and Routing Engine
  - Oracle Application Server MapViewer

ORACLE

# Coordinate Systems: Concepts

- A coordinate system (CS) is a means of assigning coordinates to a location.
  - It establishes relationships between sets of coordinates.
- All spatial data has an associated coordinate system.

ORACLE



## Coordinate Systems: Concepts

Coordinate systems are used to identify where an object is in space. This location of an object is considered absolute with respect to the coordinate system used.

Coordinate systems are also used to identify where an object is in relation to other objects. The location of objects is relative to each other within the specified coordinate system. Whenever you deal with spatial data, remember that there is a coordinate system associated with that data.

In Oracle Spatial, the coordinate system can be implicitly defined, but not specified (that is, the spatial reference ID [`SRID`] is `NULL` in `USER_SDO_GEOM_METADATA` and the `SDO_SRID` field of the geometry object is `NULL`). Alternatively, the coordinate system can be explicitly defined by setting the `SRID` column in the `USER_SDO_GEOM_METADATA` and `SDO_SRID` fields for each geometry in the layer.

# Types of Coordinate Systems

There are two types of coordinate systems:

- Georeferenced
  - Related to a specific representation of Earth
- Local or nongeoreferenced
  - Not related to any representation of Earth

Georeferenced can be of two types:

- Projected (not longitude/latitude)
- Geodetic (longitude/latitude)

ORACLE



## Types of Coordinate Systems

There are two kinds of coordinate systems. Georeferenced coordinate systems are Earth related. Local or nongeoreferenced coordinate systems are not Earth related.

Oracle Spatial supports two kinds of georeferenced data:

- Projected, where location information is specified as planar (Cartesian) coordinates that result from performing a mathematical mapping from a location on the Earth's surface to a plane. Projected information is specified in units of length such as meters or feet.
- Geodetic, where location information is expressed in angular units of longitude (which specifies positions east and west of the prime meridian) and latitude (which specifies positions north and south of the equator). In Oracle Spatial, geodetic data is expressed in decimal degrees varying from –180 degrees to 180 degrees in longitude, and from –90 degrees to 90 degrees in latitude. Lines of latitude are parallel to each other. The lengths of parallels decrease as the parallels move away from the equator until their lengths equal zero at the North Pole and the South Pole. Lines of longitude are not parallel, but converge at the poles. The distance or length specified by a degree of longitude changes from about 111 kilometers at the equator to zero at the poles.

# Spatial Query

- Is issued through SQL
- Contains a location constraint for a query window

For example:

- Find all policyholders in the projected path of a hurricane.
- Find all automated teller machines (ATMs) in my area.
  - "in my area" is the query window.

ORACLE



## Spatial Query

All spatial queries are issued through SQL. The terms "window" and "area-of-interest (AOI)" are used synonymously in this course. A window refers to the area that you are searching. An extreme case of a window query is a single window against an entire layer. The following are some spatial query examples:

- Find all policyholders in the projected path of a hurricane: "in the projected path of a hurricane" is the query window in this spatial query.
- Find all automated teller machines (ATMs) in my area. "in my area" is the query window in this spatial query.

You can issue spatial queries that have more than a single window (for example, show all the ATMs in region `A` or region `B`). Implementing spatial queries is discussed in more detail in the lesson titled "Querying Spatial Data."

# Spatial Indexing

- R-tree indexing:
  - Is based on minimum bounding rectangles (MBR)
  - Indexes two, three, or four dimensions
- Provides an exclusive and exhaustive coverage of spatial objects
- Indexes all elements within a geometry including points, lines, and polygons

ORACLE

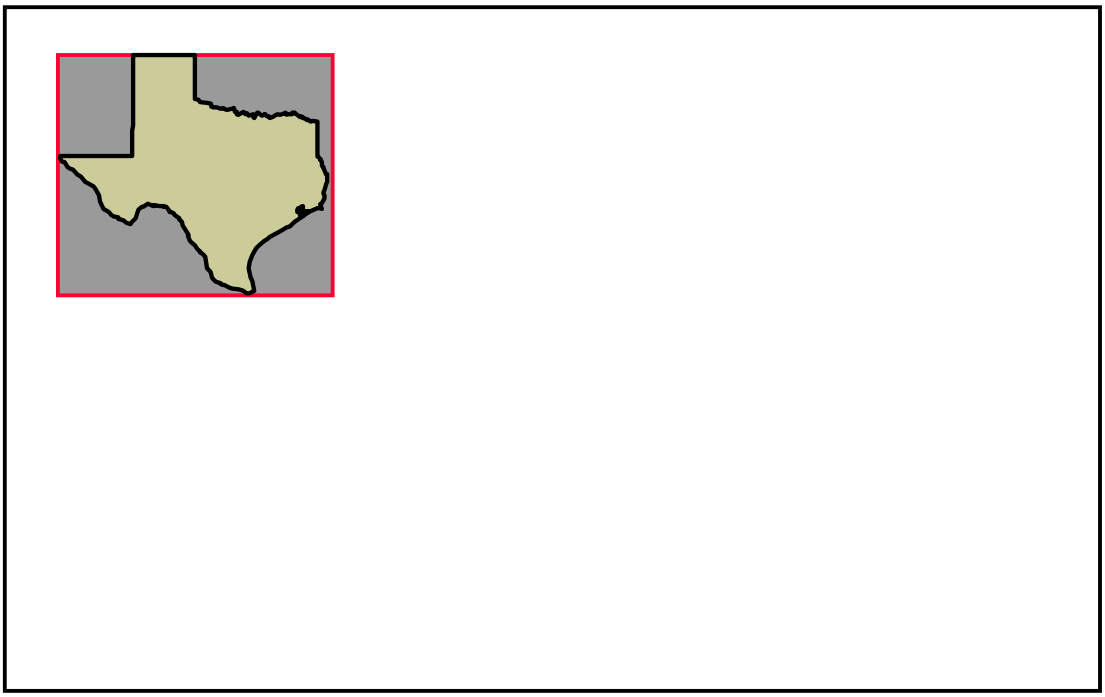


## Spatial Indexing

Oracle Spatial uses a spatial index to optimize spatial query performance. Just as an index is created on scalar data to improve query performance, a spatial index improves spatial query performance by avoiding full table scans many times.

Oracle Spatial uses R-tree indexes, which generate and store an approximation of each geometry when you create a spatial index. This approximation is generated for all types of geometries (for example, points, line strings, and polygons).

# R-Tree: Generating Minimum Bounding Rectangles (MBR)



ORACLE



## R-Tree: Generating Minimum Bounding Rectangles (MBR)

The slide example demonstrates the R-tree indexing concept. For each geometry, a single MBR is generated during index creation.

**Note:** An MBR of a point is the point itself.

# Tolerance

- Is used to associate precision with spatial data
- Reflects the distance that two points must be apart to be considered as individual points
- Must be a positive number. Smaller tolerance values result in more precise data.
- Is specified in the metadata for a spatial layer

**Tolerance = 0.05 meters**


ORACLE



## Tolerance

The tolerance in a geodetic coordinate system is defined in meters. The tolerance value for geodetic data must not be lower than 0.05 meters (5 centimeters). For nongeodetic data, the tolerance value is in the units that are associated with the coordinate system associated with the data.

For geodetic data, if tolerance is set to 0.05 meters, all the points defined in a periphery of 0.05 m from Point 1 are not considered as separate points. Point 2 is at a distance greater than 0.05 meters from Point 1, so it is considered as an individual point or location.

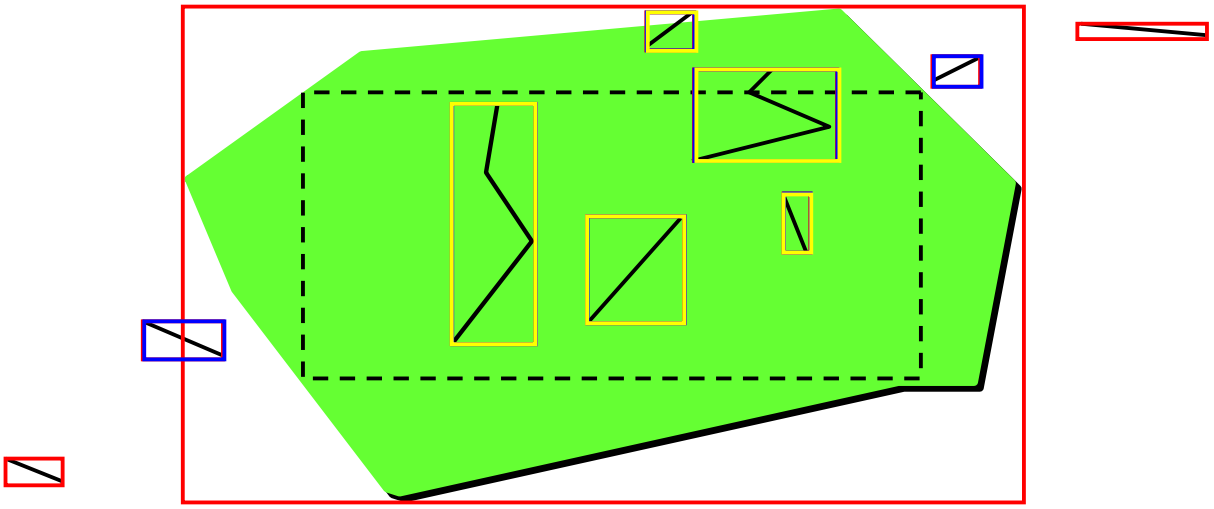
# Lesson Agenda

- Oracle Spatial and the supported geometric primitive types
- Spatial data model: Elements, geometries, and layers
- Define:
  - Coordinate systems
  - Spatial window queries and minimum bounding rectangle (MBR)
  - Spatial indexing
  - Tolerance
- Spatial query model: Primary and secondary filters
- Introduction:
  - Linear Referencing System
  - Geocoding and Routing
  - Oracle Application Server MapViewer

ORACLE

# Primary and Secondary Filters

- The primary filter compares geometry approximations, so the result is not exact.
- Interior optimizations are applied to the candidate set.
- Geometry comparisons are done only where required.



ORACLE



## Primary and Secondary Filters

Spatial uses a two-tier query model. The primary filter uses spatial indexes to compare geometry approximations and derives a superset of the result. The secondary filter may be applied on this superset to get the exact result set. Secondary filters use specific functions and operators to derive an exact result set.

In this example, the R-tree index is used to determine which geometries have any interaction with the rectangular geometry.

The R-tree spatial index is used to store approximations of every geometry stored. A minimum bounding rectangle (MBR) that covers each geometry is stored in the spatial index along with a pointer to that geometry. The query window also has an MBR. The spatial index works by comparing the approximation of the query window (MBR) with the approximations of geometries (MBRs in the index). All geometries with an MBR index that interacts with the MBR query window are returned by the spatial index query.

Note that the results of the index query (or the primary filter) are not exact because comparisons are done using approximations of each geometry. The spatial index always returns all the geometries that satisfy the spatial query, and perhaps some extra geometries.

## Primary and Secondary Filters (continued)

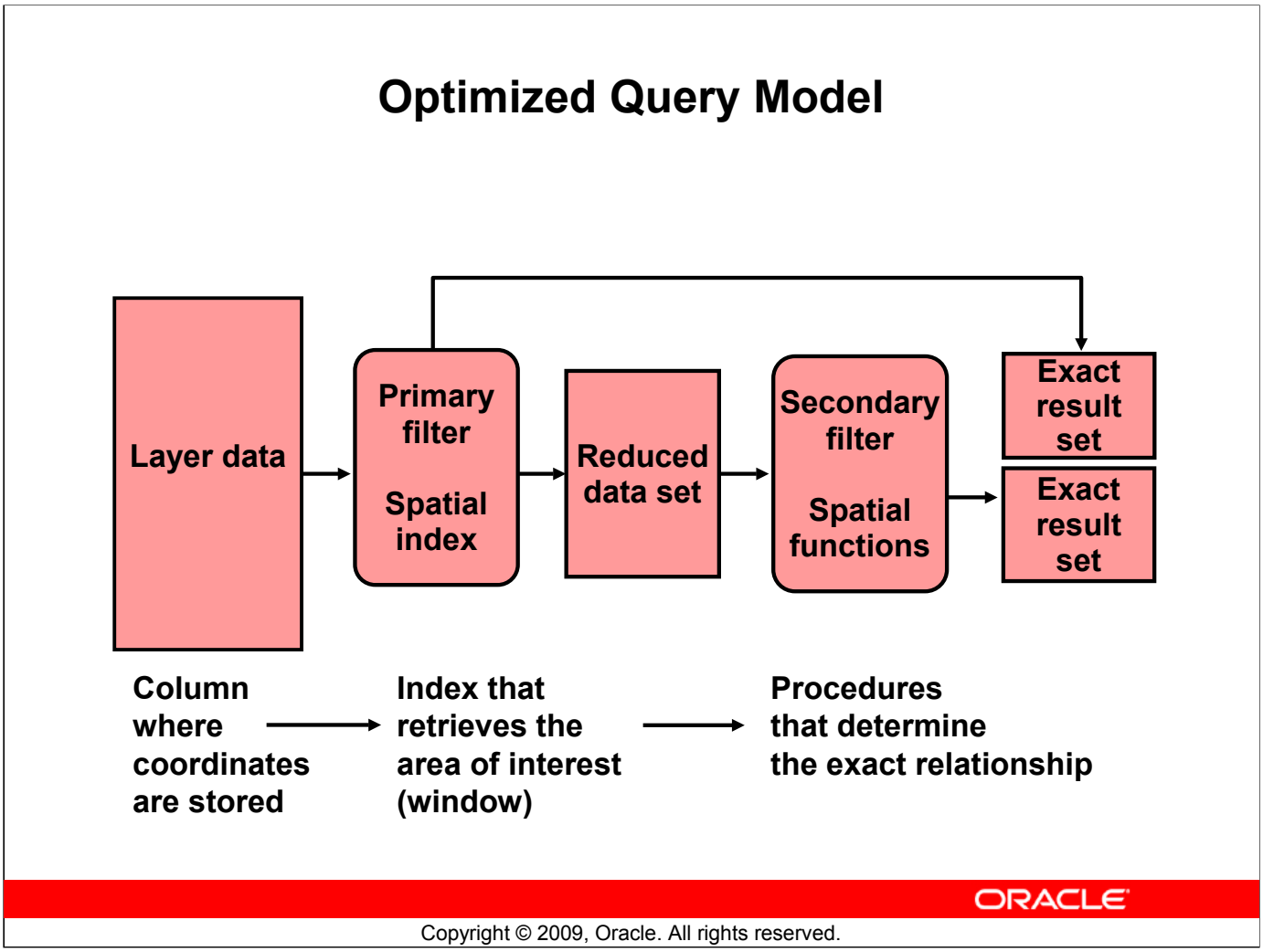
Next, interior optimizations are applied. The example shown is only for illustration purposes, and may not be exactly what Oracle Spatial does in all cases. In this case, an interior rectangle is generated, and is shown by the dotted line. Any geometry's MBR that falls completely in the interior rectangle of the county can be optimized into the result set with no true coordinate geometry calculations.

After all the optimizations are complete, Oracle Spatial performs true geometry-to-geometry comparisons, returning the last of the roads that have an interaction with the county.

A primary filter returns all the geometries that satisfy a query and possibly some additional geometries. Some applications do not always need an exact answer. For example, the `ZOOM IN/OUT` functionality of a map-display application can perform a primary filter in an Oracle database. It can then clip the results on the client to discard additional geometries returned from the primary filter.

Primary filters alone are not appropriate for every situation. For example, if you drew a circle on a viewport and asked for all the parcels that interact with the circle, an approximate answer may not be acceptable. In this scenario, you can perform a secondary filter in the Oracle server to get the exact answer.

The secondary filter process first performs a primary filter. The primary filter can automatically accept a set of geometries as interacting, even though it compares only geometry approximations. Then the coordinates of the remaining geometries that are not automatically accepted in the primary filter are examined for possible interaction. The result of a secondary filter is all the geometries that interact.




## Optimized Query Model

This slide gives you an overview of the Oracle Spatial optimized query model. Start with a spatial layer and ask yourself a spatial question. (For example, "what are the parcels that interact with a given circular area?")

Oracle Spatial has a two-stage query model: a primary filter and a secondary filter.

When a spatial index is created, geometry approximations are stored for each geometry. The primary filter portion of the query compares geometry approximations instead of the true geometries and generates a reduced data set. Comparing geometry approximations is much faster than comparing geometries.

**Note:** The optimized query model is covered in detail in the lesson titled "Querying Spatial Data."

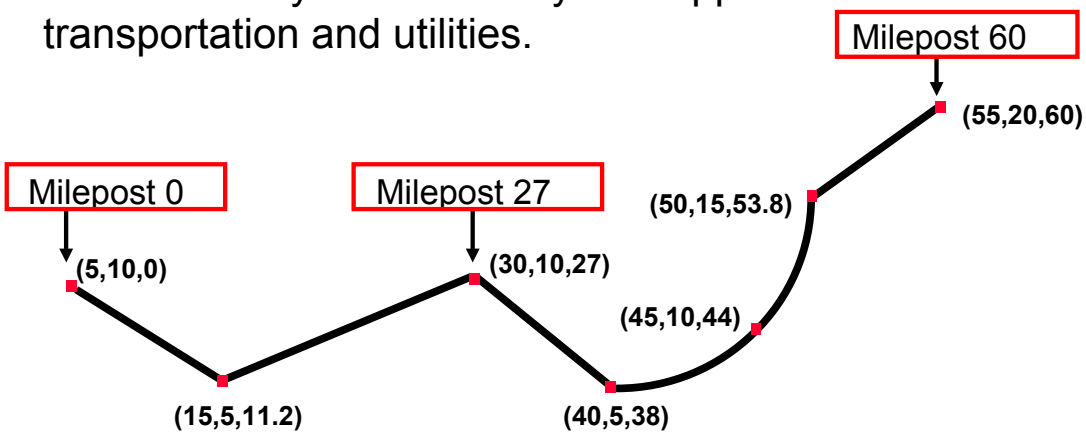
# Lesson Agenda

- Oracle Spatial and the supported geometric primitive types
- Spatial data model: Elements, geometries, and layers
- Define:
  - Coordinate systems
  - Spatial query window
  - Spatial indexing
  - Tolerance
- Spatial query model: Primary and secondary filters
- Introduction:
  - Linear Referencing System
  - Geocoding and Routing Engine
  - Oracle Application Server MapViewer

ORACLE

# Linear Referencing System (LRS)

- This is a mechanism to associate a measure value with a 2D or 3D point along a line string or a polygon.
- The measure value is typically proportional to the distance from the start measure of the geometry.
- It is commonly used in many GIS applications such as transportation and utilities.




ORACLE



## Linear Referencing System (LRS)

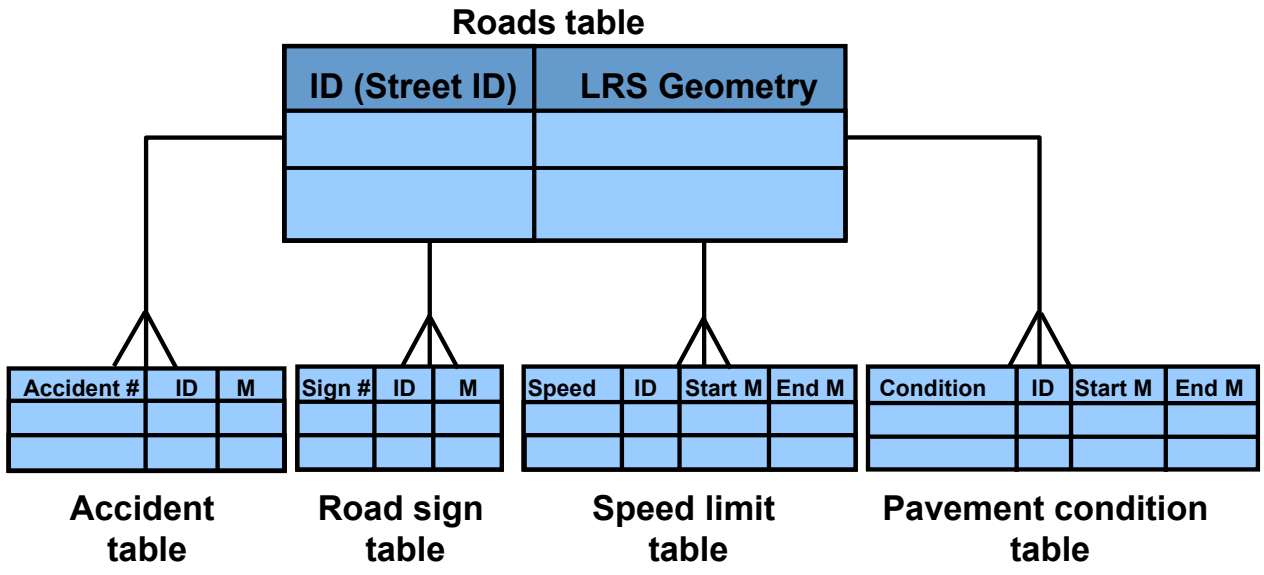
Linear Referencing Systems associate a measure value with each 2D or 3D point along a linear feature. In the example in the slide, the 2D line string has measure values that range from 0 (at the first point) to 60 (at the last point). All the measure values of points between the first and last points are between 0 and 60. Measure values must be either ascending (as in the example) or descending.

Linear referencing is commonly used in applications for transportation (roads, trains, pipelines) or logistics. A common usage is to place measure information along a highway (for example, mileposts). Linear-referenced geometries are stored in one table and the associated measure-value ranges with discrete attributes, such as speed limits, are stored in another table. For example, in the road depicted in this slide, measure values 0 through 27 could represent a 50-mile-per-hour zone and measure values 27 through 60 could represent a 35-mile-per-hour zone. It is important to note that measure ranges that represent speed limits do not have to begin and end on geometry points (that is, a section of a highway from measure 5 to 15 could represent a 25-mile-per-hour zone).

Measure values are typically proportional to the distance from the start measure of the geometry, but not always. In the example in the slide, the point at (X,Y) = (30,10) is almost the line string midpoint, and its measure value 27 is almost the measure midpoint. If from (5,10) through (30,10), the road segment had a very steep incline, the measure value at (30,10) could be set to 45 to account for the slope.

**Note:** You learn about LRS in detail in the lesson titled “Implementing a Linear Referencing System.”

# LRS Application: Transportation Application in the Database






## LRS Application: Transportation Application in the Database

This is a simple data model for a transportation application. The linear-referenced roads are each stored once in the Roads table. Each road can be associated with many accidents, road signs, speed limits, and pavement conditions. The Roads table has a one-to-many relationship with each of its associated attribute tables.

# Geocoding

- Process of associating latitude and longitude coordinates with postal addresses
- Used for:
  - Location or proximity queries
  - Demographic analysis
  - Business finders
  - Routing and directions
  - Mapping

ORACLE



## Geocoding

Geocoding takes a textual address as input and returns a longitude/latitude coordinate associated with the address. Reverse geocoding is also supported by Oracle Spatial. Reverse geocoding takes longitude and latitude information as input to get the associated postal address.

You learn more about geocoding in the lesson titled “Geocoding Address Data.”

# Routing Engine

- Provides routing information such as driving distances and directions between two locations
- Provides routing information about multiple routes with the same start location and different end locations
- Enables hosting of XML-based Web services to provide routing information

ORACLE



## Routing Engine

This slide introduces you to the concept of routing in the Oracle Spatial technology. You learn about this in detail in the lesson titled "Using the Spatial Routing Engine."

# Oracle Application Server MapViewer

- Is used for map rendering and visualization of spatial data stored in either Oracle Spatial or Oracle Locator
- Is a component of the Oracle Application Server
- Stores style, theme, and map information in the Oracle database as well as the data to be rendered
- Can call the XML API via any language that communicates with the Oracle Application Server (HTML forms, Java, C, OCI, .NET, and so on):
  - AJAX-based Javascript API
  - Java API, JSP Tag library, and OGC WMS API
- Can render maps for Oracle 8.1.7 and later

ORACLE



## Oracle Application Server MapViewer

Oracle Application Server MapViewer is a programmable tool for rendering simple maps by using the spatial data managed by Oracle Spatial. MapViewer is a part of Oracle's application server (both Standard and Enterprise editions), and includes a component that performs cartographic rendering and a component that is used to define and manage map metadata and portrayal information.

All spatial data and map information (including map definitions and style information) are stored in the Oracle database.

MapViewer renders images and is called using Extensible Markup Language (XML) from applications that interface with Oracle Application Server. MapViewer can either stream images to a client or return the URL of the rendered map.

# Summary

In this lesson, you should have learned how to:

- Describe the supported geometric primitive types
- Define the Oracle spatial data model
- Describe the spatial query model
- Describe the concept of spatial indexing
- Describe coordinate systems
- Define spatial queries
- Describe Linear Referencing System (LRS)
- Explain geocoding and routing engine
- Describe Oracle Application Server MapViewer

ORACLE



## Summary

This lesson provides an overview of all the concepts covered in this course. All concepts are discussed in more detail in later lessons.

# Practice 1: Overview

This practice includes a few questions to recapitulate the Oracle Spatial concepts and terminologies.

ORACLE



## Practice 1: Overview

In this practice, you answer some questions to recapitulate the Oracle Spatial concepts and terminologies.

## Practice 1

1. Answer "true" or "false" for the following statements about the Points geometry:
   i. A 2D point defines a location in the X and Y coordinates.
   ii. A point may be used to represent the location of a building or an automated teller machine (ATM).
2. Answer "true" or "false" for the following statements about the Line Strings geometry:
   i. Self-crossing line strings are invalid.
   ii. When line strings close to form a ring, they have an area.
   iii. The boundary of a line string is defined by its end points.
3. Answer "true" or "false" for the following statements about the Polygons geometry:
   i. Polygons have an area.
   ii. Self-crossing polygons are valid and supported in Oracle Spatial.
4. Define an optimized rectangle.
5. Answer whether the statement is true or false:
   i. An outer ring polygon is defined in the counterclockwise direction.
   ii. Inner rings must be followed by their outer ring.
   iii. A geometry must be composed of the same element types.
   iv. An Oracle table's column of the `SDO_GEOMETRY` type is defined as a spatial layer.
   v. A spatial layer can contain different types of geometries.
   vi. Tolerance defines the precision of spatial data.
6. Fill in the blanks:
   i. A rectangle that minimally encloses a geometry is called a_____.
   ii. An optimized circle is a polygon defined by_____distinct points on the circumference of the circle.
   iii. The primary filter uses the _____ to compare geometry approximations to get a superset of the result.
7. Match the following:

| Term | Definition |
|---|---|
| a. Geocoding | a. Associates a measure value with each 2D or 3D point along a linear feature |
| b. Linear Referencing System | b. Provides routing information such as driving distances or directions between two locations |
| c. Routing | c. Associates latitude and longitude coordinates with postal addresses |

# 2 Creating Spatial Layers

# Objectives

After completing this lesson, you should be able to do the following:

- Describe the schema associated with Oracle Spatial
- Explain how spatial data is stored using the Oracle Spatial object model
- Create a table with a spatial layer
- Insert different types of geometries into a spatial layer
- Update the `USER_SDO_GEOM_METADATA` view with metadata information about spatial layers

ORACLE



## Objectives

In this lesson, you are introduced to Oracle's native data type for Spatial, `SDO_GEOMETRY`. You also learn to create a table with a spatial layer, insert different types of geometries, and update the spatial metadata.

# Lesson Agenda

- MDSYS schema and spatial data management
- Spatial native data type: SDO_GEOMETRY
- Geometry elements
- Construction of geometries by using the INSERT statement
- Spatial metadata

ORACLE

# MDSYS Schema

- Is created when Oracle Spatial or Locator is installed
- Is the owner of the Spatial types, packages, functions, procedures, and metadata
- Is the privileged user with the ADMIN role
- Is locked by default
- Must not be accessible by typical users
  - Never create data in this schema.

ORACLE



## MDSYS Schema

The MDSYS privileged user is created when Oracle Spatial or Oracle Locator is installed. The MDSYS schema owns the storage, syntax, and semantics of the supported geometric data types. This user is similar to the SYS and SYSTEM users in that it is a privileged account with the ability to grant privileges (GRANT) to other users on the system. There should be no need for users to log in to this account. You must never create any data or objects as the MDSYS user. Because this is a privileged user account, care should be taken to leave it as a locked account.

## Typical Steps for Spatial Data Management

1. Create a table to store spatial data.
2. Insert spatial data as bulk loads or incrementally.
3. Update the spatial metadata view.
4. Validate and debug geometries if required.
5. Create spatial indexes.
6. Run spatial queries.

ORACLE



### Typical Steps for Spatial Data Management

The steps mentioned in the slide are the basic tasks that you typically perform to get started with any kind of spatial data. You must define an Oracle table with one or more `SDO_GEOMETRY` columns to store geometrical data. This table must have some nonspatial attributes that have a one-to-one relationship with the `SDO_GEOMETRY` column.

You can insert rows in this table by using the `INSERT` statement. There are also ways to bulk-load spatial data into Oracle tables. It is also essential to validate the geometries before using them in spatial queries. Oracle Spatial provides various validation and debugging functions that can help remove invalid geometries and also fix some discrepancies. Spatial analysis results are guaranteed only when the geometries are valid.

You must also update the metadata views with dimensional information about the geometries. Create spatial indexes to enhance the performance of spatial queries. Finally, you can run your spatial queries.

# Lesson Agenda

- MDSYS schema and spatial data management
- Spatial native data type: SDO_GEOMETRY
- Geometrical elements
- Construction of geometries by using the INSERT statement
- Spatial metadata

## Spatial Native Data Type: SDO_GEOMETRY

- A geometry is stored as an object, in a single row, in a column of the SDO_GEOMETRY type.
- SDO_GEOMETRY is defined as an object type.
- The SDO_GEOMETRY object type has methods that provide access to its attributes.

ORACLE



### Spatial Native Data Type: SDO_GEOMETRY

Oracle Spatial defines an object type, SDO_GEOMETRY, which can store a geometry definition. You can use the DESCRIBE command to get the structure of this SDO_GEOMETRY object type. The SDO_GEOMETRY object type has some attributes, overloaded constructors, and member methods that provide access to its attributes. Some of its attributes are based on native data types, whereas some others are based on different object types.

You learn about them in detail in the following slides.

**Note:** Run the DESCRIBE SDO_GEOMETRY command to view the structure of this native data type.

## SDO_GEOMETRY Object

- SDO_GEOMETRY object:

```
SDO_GTYPE          NUMBER
SDO_SRID           NUMBER
SDO_POINT          SDO_POINT_TYPE
SDO_ELEM_INFO      SDO_ELEM_INFO_ARRAY
SDO_ORDINATES      SDO_ORDINATE_ARRAY
```

- Example:

```
CREATE TABLE city (
      city_name    VARCHAR2(30),
      totpop       NUMBER(9),
      geom         SDO_GEOMETRY);
```

ORACLE



### SDO_GEOMETRY Object

The SDO_GEOMETRY object has five fields. Each field is discussed in detail in the next slides. Two of the fields are of the NUMBER type and the other three fields are of the object types.

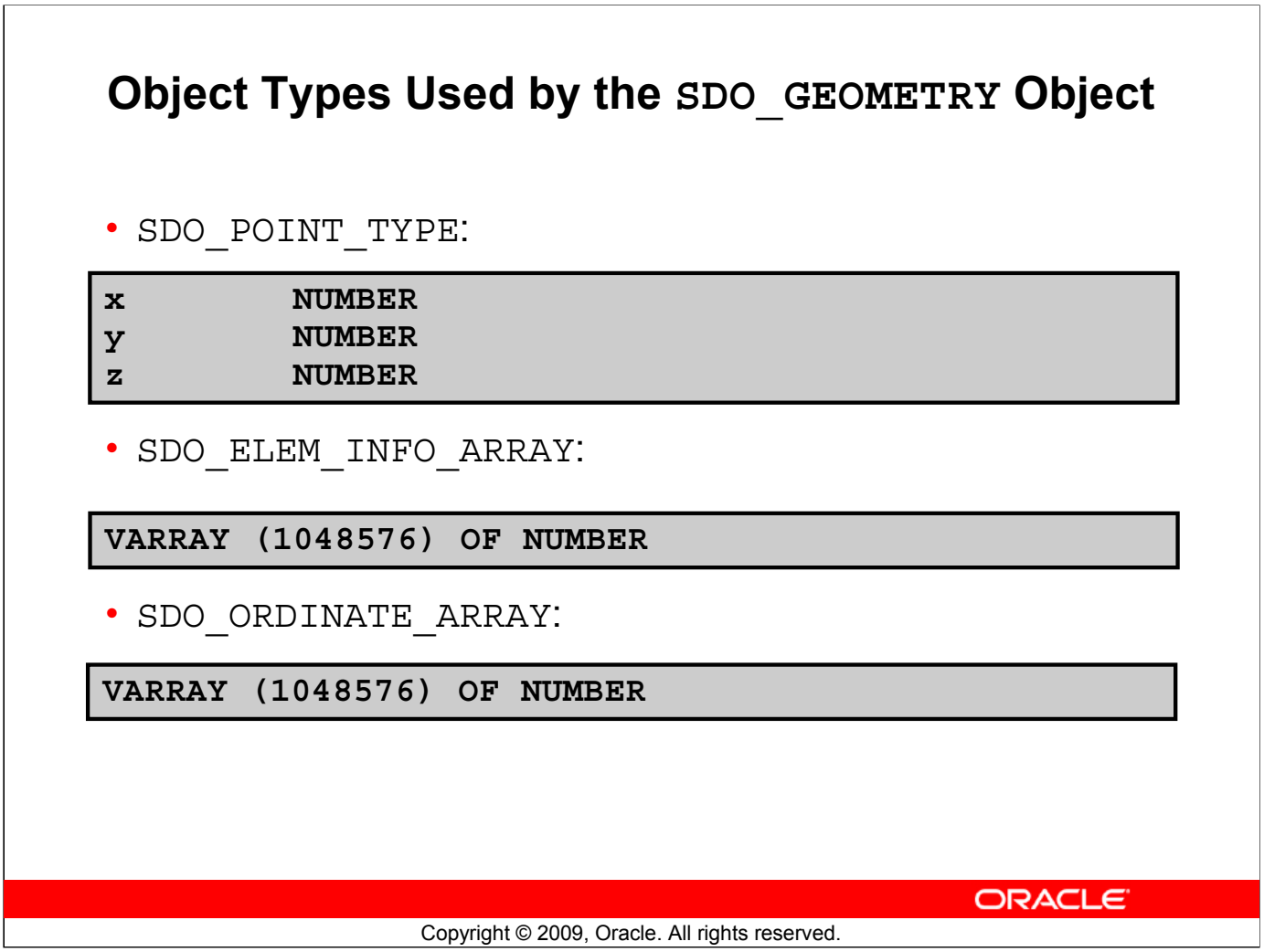
The SDO_GEOMETRY column is accessed through SQL and works with all Oracle utilities: Data Pump files (that is, expdp and impdp), SQL*Loader, and the earlier database import and export utilities.

You create tables with the columns of the SDO_GEOMETRY type in the same way that you create tables with other Oracle native data types. There is no such thing as a spatial table, only a table with one or more spatial columns of the SDO_GEOMETRY type. Spatial tables are ordinary Oracle tables, with ordinary storage clauses that contain one or more SDO_GEOMETRY columns. The SDO_GEOMETRY object contains an entire geometry in a single-row–single-column pair of an Oracle table.

The city table has the following nonspatial columns: city_name and totpop, which have a one-to-one relationship with the SDO_GEOMETRY column, geom.

The data modeling concept for normalized tables even applies to tables with the SDO_GEOMETRY columns.

**Note:** When creating a column of the SDO_GEOMETRY type in databases before Oracle Spatial 10*g*, the type must be specified as MDSYS.SDO_GEOMETRY.




## Object Types Used by the SDO_GEOMETRY Object

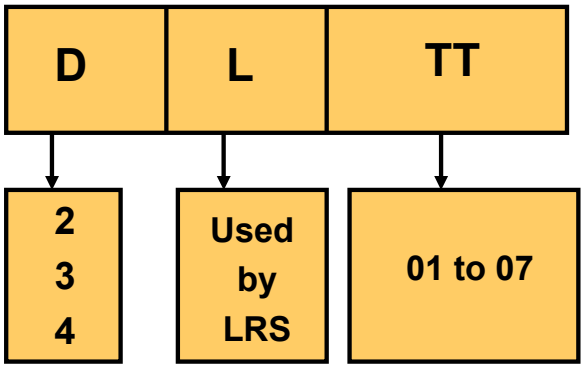
The SDO_POINT_TYPE object is a field within the SDO_GEOMETRY object. It is used to store a single point of up to three dimensions. The SDO_POINT_TYPE object must not be used outside the SDO_GEOMETRY object (that is, do not create tables with a column of the SDO_POINT_TYPE type).

SDO_ELEM_INFO_ARRAY is a field within the SDO_GEOMETRY object. It describes the elements stored in this geometry. This field is discussed in detail in a subsequent slide.

SDO_ORDINATE_ARRAY is a field within the SDO_GEOMETRY object. It contains all the ordinates for all the elements that make up the geometry. This is described in detail in subsequent slides.

## SDO_GTYPE Field

- Defines the type of geometry stored in the object
- Is specified in the DLTT format where:
  - D identifies the number of dimensions
  - L identifies the linear referencing measure dimension
  - TT identifies the geometry type




ORACLE



### SDO_GTYPE Field

This is the first of the five fields in the SDO_GEOMETRY object. As discussed earlier, a geometry can be made up of one or more elements. You can think of SDO_GTYPE as the "big picture." It defines what kind of geometry is being stored in the SDO_GEOMETRY object.

The SDO_GTYPE is specified in the DLTT format as explained in the slide. The value of the D component can be either 2, 3, or 4, which represents the number of ordinates that are used to define a vertex. The value of L is used to indicate the Linear Referencing System (LRS) measure position that is explained in detail later. The values ranging from 01 to 07 can be used to indicate the geometry type. For example, a value of 02 indicates a line string that can contain straight lines or arcs.

An SDO_GTYPE value of 2003 indicates that the geometry is a two-dimensional polygon without any linear referencing.

You must associate the most appropriate SDO_GTYPE with your geometry. For example, a single polygon element can be categorized as SDO_GTYPE 7, but, optimally, it is categorized as SDO_GTYPE 3. Note that SDO_GTYPE 4 (heterogeneous collection) must be used only if there is a requirement for that geometry to store different element types. Heterogeneous collections are not part of the Open Geospatial Consortium (OGC) standard, and may not be supported by all applications.

**Note:** TT values of 08 and 09 are used to represent solids and multisolids, which are not covered in this course.

## SDO_GTYPE Values with Dimensionality

| SDO_GTYPE | Four-digit GTYPEs: Include dimensionality | |
|---|---|---|
| | 2D | 3D |
| 1. POINT | 2001 | 3001 |
| 2. LINE STRING | 2002 | 3002 |
| 3. POLYGON | 2003 | 3003 |
| 4. HETEROGENEOUS COLLECTION | 2004 | 3004 |
| 5. MULTIPOINT | 2005 | 3005 |
| 6. MULTILINE STRING | 2006 | 3006 |
| 7. MULTIPOLYGON | 2007 | 3007 |

ORACLE



### SDO_GTYPE Values with Dimensionality

The dimensionality of the geometry, which determines how many numbers make up a point, is encoded into SDO_GTYPE as the first of the four digits.

## SDO_SRID Field

- The SDO_SRID field indicates the coordinate system associated with the geometry.
- If set to NULL, no coordinate system is associated with the geometry.
- The CS_SRS dictionary view records all the supported coordinate systems.



### SDO_SRID Field

This is the second field in the SDO_GEOMETRY object. This is used to associate spatial data with a coordinate system. Oracle Spatial has a dictionary view called CS_SRS, which lists all the supported coordinate systems. The value set in the SDO_SRID field must match a value in the spatial reference ID (SRID) column in the CS_SRS view, or be NULL.

## SDO_POINT Field

- Is the optimized space for storing point geometries
- Is ignored if SDO_ELEM_INFO and SDO_ORDINATES are not NULL
- Is used when the geometry contains a single point
- Is defined using the SDO_POINT_TYPE object type
  - Do not use SDO_POINT_TYPE outside the SDO_GEOMETRY object.

ORACLE



### SDO_POINT Field

This is the third field in the SDO_GEOMETRY object. This is an optimized way to store a single point in the SDO_GEOMETRY object. If a single two- or three-dimensional point is being stored, place it in the SDO_POINT field of the SDO_GEOMETRY object. The SDO_POINT field provides optimal storage and performance. The value stored in the SDO_POINT field is ignored if the SDO_ELEM_INFO and SDO_ORDINATES fields are not null.

The SDO_POINT attribute is defined using the SDO_POINT_TYPE object type, which has the attributes x, y, and z, all of which are of the NUMBER type. The SDO_POINT field must be used within the context of the SDO_GEOMETRY object (Oracle Spatial does not work with a column in a table defined as SDO_POINT_TYPE).

# Element: Point

- Load the SDO_POINT field to optimize point data storage.
- To generate a spatial index on the SDO_POINT field, SDO_ELEM_INFO and SDO_ORDINATES must be NULL.
- If SDO_ELEM_INFO and SDO_ORDINATES are not NULL, SDO_POINT is ignored by Oracle Spatial (not spatially indexed).

```
INSERT INTO TELEPHONE_POLES
  VALUES (attribute_1, …, attribute_n,
       SDO_GEOMETRY (
       3001, null,
         SDO_POINT_TYPE (-75.2,43.7,200),
             null, null)
 );
```

ORACLE



## Element: Point

This is an example of storing a single point in the SDO_POINT field. The SDO_POINT field optimizes storage for a single point. You can store x, y, and z values in the SDO_POINT field.

To generate a spatial index in the SDO_POINT field, the SDO_ELEM_INFO and SDO_ORDINATES fields must be null. The SDO_POINT field is ignored by Oracle Spatial if the SDO_ELEM_INFO and SDO_ORDINATES fields are not null.

The example in the slide shows you the use of an SDO_GEOMETRY constructor (highlighted), which is the syntax that is specific to object types. It can be thought of as a logical cast operation, which forces the data to the type specified. Also, you must use the SDO_POINT_TYPE constructor to instantiate an object of the SDO_POINT type.

**Note:** If you store a 2D point in the SDO_POINT field, set the z value to NULL. For example:

```
INSERT INTO TELEPHONE_POLES VALUES
(attribute_1, attribute_n,
 SDO_GEOMETRY (2001, null,
  SDO_POINT_TYPE(-75.2,43.7,NULL), null, null));
```

## SDO_ELEM_INFO Field

- The SDO_ELEM_INFO field defines how to interpret the ordinates stored in the SDO_ORDINATES attribute.
- Numeric entries in the varray are considered in groups of three.
- The triplet values stored in the array are:
  - Ordinate offset: Position of the first ordinate of an element in the SDO_ORDINATES array. It starts from 1.
  - Element type: Type of element (point, line, polygon, and so on)
  - Interpretation: Straight lines, circular arc, or header

ORACLE



### SDO_ELEM_INFO Field

This is the fourth field in the SDO_GEOMETRY object. A geometry contains one or more elements. If a single two- or three-dimensional geometry is stored in the SDO_GEOMETRY column, a description of each element that makes up the geometry is stored in the SDO_ELEM_INFO field. The SDO_ELEM_INFO field is a VARRAY (that is, a varying-sized array) of the NUMBER type. Entries in the SDO_ELEM_INFO field must be considered in groups of three numbers. The triplet values represent:

- **Ordinate offset:** Position of the first ordinate of an element in the SDO_ORDINATES array. The index value corresponds to the first ordinate of the element that the triplet describes. The first element of a geometry always begins at position 1.
- **Element type:** Type of the element (point, line, polygon, and so on)
  - Simple elements: 1, 2, 1003, 2003
  - Compound elements: 4, 1005, 2005
  - All possible values for the element type and their interpretations are discussed in the remaining portion of this lesson.
- **Interpretation:**
  - Interpretation takes on a different meaning depending on the value of the element type.
  - It can mean that all points for this element are connected with straight lines or circular arcs, or it might mean that this value is a header for a compound element. This is discussed in detail in the subsequent slides.

## SDO_ORDINATES Field

- The SDO_ORDINATES field stores the ordinates that make up the geometry elements.
- Elements stored in SDO_ORDINATE_ARRAY are interpreted using the values in the SDO_ELEM_INFO field.
- A four-vertex two-dimensional polygon is stored as (X1,Y1,X2,Y2,X3,Y3,X4,Y4).
- A four-vertex three-dimensional polygon is stored as (X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3,X4,Y4,Z4).

ORACLE



### SDO_ORDINATES Field

This is the fifth field in the SDO_GEOMETRY object. This field is a VARRAY of the NUMBER type. It contains all ordinates that make up the elements described by the SDO_ELEM_INFO field.

**Note:** Both the SDO_ORDINATES and SDO_ELEM_INFO fields are defined as VARRAYs whose maximum length is 1,048,576. In the case of SDO_ELEM_INFO, this means that the maximum number of simple elements that any geometry can have is 349,525 (1,048,576/3). In the case of SDO_ORDINATES, the most two-dimensional coordinates that any geometry can have is 524,288. If the geometries are three-dimensional (or two-dimensional with an LRS measure), the maximum number of coordinates is 349,525. If there are four ordinates that are stored for each point in the geometry, the maximum number of coordinates is 262,144.

# Element Types: Summary

| VALUE | Element Type | Interpretation Value |
|---|---|---|
| 1 | POINT | Number of points in the collection; or<br>0: Oriented point |
| 2 | LINE STRING | 1: Straight lines<br>2: Circular arcs |
| 3<br>1003<br>2003 | POLYGON<br>(Outer)<br>(Inner) | 1: Straight lines<br>2: Circular arcs<br>3: Optimized rectangle<br>4: Circle |
| 4 | COMPOUND LINE STRING | Number of type 2 subelements that make up the line string |
| 5<br><br>1005<br>2005 | COMPOUND POLYGON<br><br>(Outer)<br>(Inner) | Number of type 2 subelements that make up the polygon |

ORACLE



## Element Types: Summary

So far, you have seen how to store data in the `SDO_POINT` field of the `SDO_GEOMETRY` object. Now you start looking at storing elements in the `VARRAY` fields of the `SDO_GEOMETRY` object. This slide summarizes how to represent an element in the `SDO_ELEM_INFO` field. This slide is repeated after the element examples, and should be easier to understand the next time you see it.

As stated earlier, a geometry can be made up of one or more elements. If the geometry is not a single point stored in the `SDO_POINT` field, each element has a triplet value entry in the `SDO_ELEM_INFO` array (Ordinate Offset, Element Type, Interpretation).

Do not confuse element types (Element Type) with `SDO_GTYPE`s. `SDO_GTYPE`s form the "big picture," which represents the type of geometry that is stored. A geometry is made up of one or more elements, and the element type describes the element.

This slide does not show ordinate offsets. The ordinate offset is a number that represents where the element begins in the `SDO_ORDINATES` array. This slide describes only the other two parts of the triplet value (Element Type, Interpretation).

## Element Types: Summary (continued)

The element type 3 is replaced with either 1003 (outer ring) or 2003 (void), and the element type 5 is replaced with either 1005 (outer ring) or 2005 (void). The four-digit element type for polygons was introduced to make it easier to determine whether a polygon element is an outer ring or a void:

- **Element type 1:** For storing points or multipoints. The interpretation field describes how many points are stored.
- **Element type 2:** For storing line strings. Interpretation 1 represents a conventional line string connected with straight lines. Interpretation 2 represents a circular arc line string made up of one or more circular arcs.
- **Element type 3:** In a four-digit format, 1003 represents an outer ring polygon and 2003 represents an inner ring (or void):
  - **Interpretation 1:** Conventional polygon, connected with straight lines
  - **Interpretation 2:** Arc polygon, connected with circular arcs
  - **Interpretation 3:** Optimized rectangle, which requires only lower-left and upper-right points to represent the rectangle
  - **Interpretation 4:** Optimized circle (not truly a polygon, but categorized with polygons because it has an area)
- **Element type 4:** Compound line string, which is a contiguous element that contains straight lines and circular arcs. The interpretation field determines how many subelements (that is, triplet values in the `SDO_ELEM_INFO` field) make up this compound line string.
- **Element type 5:** 1005 represents an outer ring compound polygon (a contiguous element that contains straight lines and circular arcs). 2005 represents a compound polygon that is an inner ring (or void). The interpretation field determines how many subelements (that is, triplet values in the `SDO_ELEM_INFO` field) make up this compound polygon.

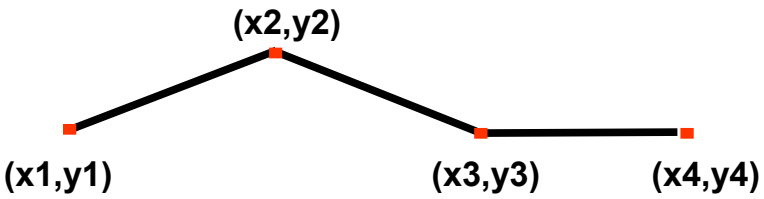
# Lesson Agenda

- MDSYS schema and spatial data management
- Spatial native data type: SDO_GEOMETRY
- Geometry elements
- Construction of geometries by using the INSERT statement
- Spatial metadata

ORACLE

# Element: Line String

- Each line segment is defined by two points.
- The last point from one segment is the first point of the next segment.
- Line segments must be contiguous.
- Line strings that close to form a ring have no implied area.
- Line segments can cross each other.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 2 | 1 |



## Element: Line String

In the table in the slide is the triplet value (Ordinate Offset, Element Type, Interpretation) that you store in the `SDO_ELEM_INFO` field. Most of the ordinate offsets that you see in the subsequent examples start at 1 because the examples depict geometries with a single element. There are some examples with geometries that have more than one element, and, in those cases, you see ordinate offsets greater than 1.

The element type 2 corresponds to a line string element. A line string element is made up of one or more contiguous segments.

The interpretation for an element type 2 defines how to connect the segments that make up the line string. In this example, the interpretation is 1 and it corresponds to a line string whose segments are connected with straight lines (a conventional line string).

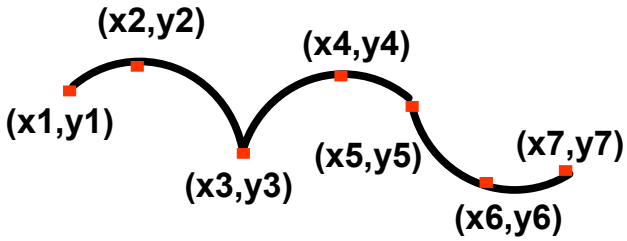
Line strings that close and form a ring have no implied area.

Line segments can cross each other.

**Note:** `SDO_GTYPE` for this geometry is 2002 (a single line string element). Note that `SDO_GTYPE` is the first field of the `SDO_GEOMETRY` object and describes the entire geometry (the big picture).

# Element: Arc String

- Each arc is defined by three distinct points on the circumference of a circle.
- The last point from one arc is the first point of the next arc.
- Arcs must be contiguous.
- Arcs that close to form a ring have no implied area.
- Arcs can cross each other.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 2 | 2 |

ORACLE



## Element: Arc String

In the table in the slide is the triplet value (Ordinate Offset, Element Type, Interpretation) that you would store in the `SDO_ELEM_INFO` field.

The element type 2 corresponds to a line string element. A line string element is made up of one or more contiguous segments.

The interpretation defines how to connect the segments that make up the line string. In this example, the interpretation is 2, and it corresponds to a line string whose segments are connected with circular arcs. A circular arc is defined by three distinct points on the circumference of a circle: the first point, the last point, and any distinct point in between. The last point of one circular arc is the first point of the next circular arc.

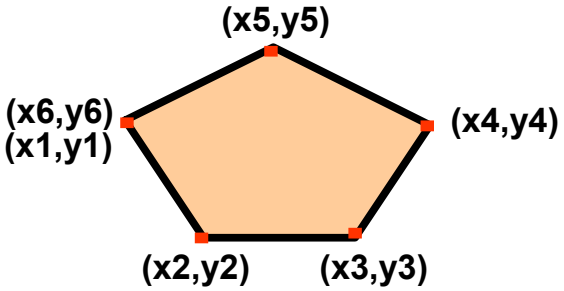
Arc strings that close and form a ring have no implied area.

Arcs can cross each other.

**Note:** `SDO_GTYPE` for this geometry is 2002 (a single line string element).

## Element: Polygon

- The element type 1003 indicates that it is an outer ring polygon.
- Interpretation 1 means that the vertices are joined by straight lines.
- The last coordinate equals the first coordinate.
- It is stored in the counterclockwise rotation.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1003 | 1 |

ORACLE



### Element: Polygon

In general, the element type 3 corresponds to a polygon. Instead of specifying an element type 3, it is recommended to clearly identify whether a polygon is an outer ring or an inner ring (or void). Outer ring polygons are represented as the element type 1003, and inner ring (void) polygons are represented as element type 2003.

Outer ring polygons must be stored with a counterclockwise rotation. Inner ring (void) polygons must be stored with a clockwise rotation. Enforcing the rotation enables faster area calculations. When determining the rotation, the lower left is where the lowest values of the coordinate system are.

The last point of a polygon must be the same as the first point. Even though this point could be derived, it must be repeated to conform to the Open Geospatial Consortium (OGC) standard.

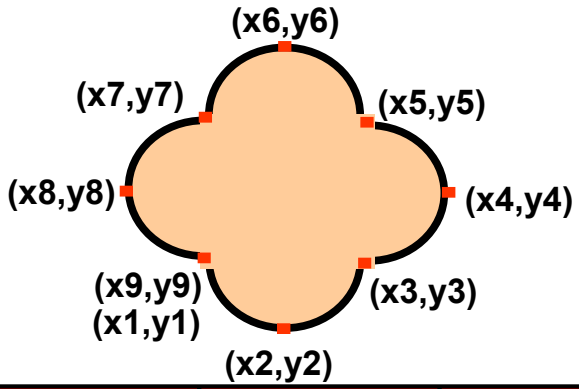
The interpretation defines how to connect the segments that make up the polygon. In this example, the interpretation is 1, and it corresponds to a polygon whose segments are connected with straight lines (a conventional polygon).

Line segments of polygons cannot cross each other.

**Note:** `SDO_GTYPE` for this geometry is 2003 (a single polygon element).

## Element: Arc Polygon

- The element type 1003 indicates that it is an outer ring polygon.
- Interpretation 2 indicates that the line segments are circular arcs.
- It is stored in the counterclockwise rotation.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1003 | 2 |

ORACLE



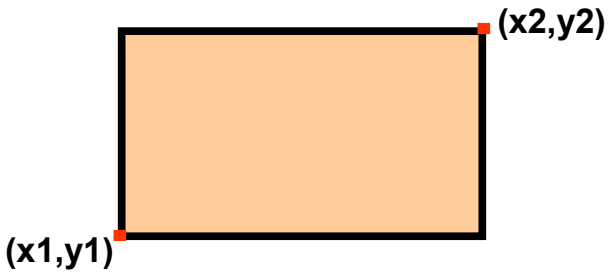
### Element: Arc Polygon

Interpretation defines how to connect the segments that make up the polygon. In this example, the interpretation is 2, and it corresponds to a polygon whose segments are connected with circular arcs. A circular arc is defined by three points on the circumference of a circle.

**Note:** `SDO_GTYPE` for this geometry is 2003 (a single polygon element).

# Element: Rectangle

- The element type 1003 indicates that it is an outer ring polygon.
- Interpretation 3 indicates that it is an optimized rectangle.
- It is defined by a lower-left point and an upper-right point.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1003 | 3 |

ORACLE



## Element: Rectangle

The element type 1003 corresponds to an outer ring polygon.

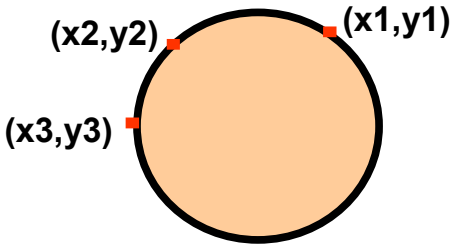
If the polygon is an inner ring (void) polygon, it is represented as the element type 2003.

In this example, the interpretation is 3, and it corresponds to a rectangle optimized for storage. Only the lower-left and upper-right points of the rectangle are stored. An optimized rectangle has an area.

**Note:** `SDO_GTYPE` for this geometry is 2003 (a single polygon element).

# Element: Circle

- Interpretation 4 indicates that it is an optimized circle.
- It is defined by any three distinct points on the circumference.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1003 | 4 |

ORACLE



## Element: Circle

The element type 1003 indicates an outer ring polygon.

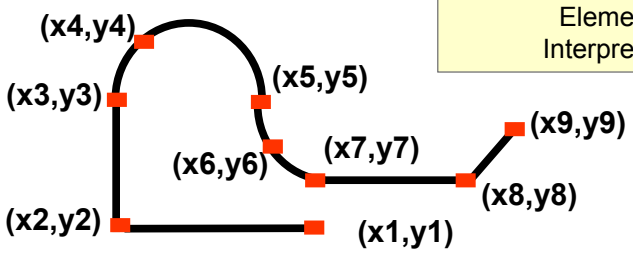
In this example, the interpretation is 4, and it corresponds to a circle. The circle is represented by any three distinct points on the circumference.

**Note:** `SDO_GTYPE` for this geometry is 2003 (a single polygon element).

# Element: Compound Line String

| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 4 | 3 |
| 1 | 2 | 1 |
| 5 | 2 | 2 |
| 13 | 2 | 1 |

The first triplet header defines the subelements.
Element type 4: Compound line string
Interpretation 3: Number of subelements




| Ordinate offset: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ordinate array: | x1 | y1 | x2 | y2 | x3 | y3 | x4 | y4 | x5 | y5 | x6 | y6 | x7 | y7 | x8 | y8 | x9 | y9 |

ORACLE



## Element: Compound Line String

The element type 4 corresponds to a compound line string. Line strings discussed earlier (element type 2) are either all connected with straight lines or all connected with circular arcs. With a compound line string, you can define a single contiguous line string element that is made up of straight lines and circular arcs.

Compound line strings have the same properties as a line string. Self-crossing lines are supported and no area is ever implied, even if it closes to form a ring.

A compound line string is defined by a series of contiguous subelements. Each subelement is a straight line or a circular arc. The first triplet is a header. The interpretation in the header denotes how many subelements (or triplets in the `SDO_ELEM_INFO` field) make up this compound line string.

All the subelements must be of the element type 2 (interpretation 1 or 2) and must be contiguous (that is, the last point of a subelement is the first point of the next subelement).

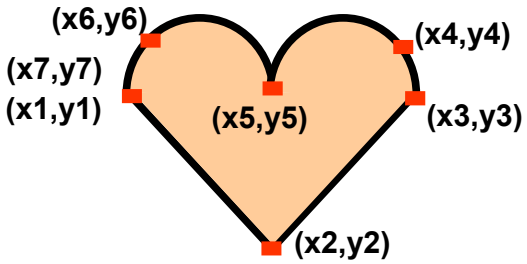
In the example in the slide, notice that the ordinate offsets of each subelement are different and correspond to where the subelement begins in the ordinate array.

**Note:** `SDO_GTYPE` for this geometry is 2002 (a single line string element).

# Element: Compound Polygon

The first triplet header defines the subelements.
Element type 1005: Outer ring compound polygon
Interpretation 2: Number of subelements

| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1005 | 2 |
| 1 | 2 | 1 |
| 5 | 2 | 2 |




ORACLE



## Element: Compound Polygon

The element type 5 corresponds to a compound polygon. Compound polygons were not added to make the product more complex. A compound polygon is often the result of a buffer operation. If you buffer a line, the result is a compound polygon (that is, straight lines on either side of the line that you buffered and half circles around the end points).

A compound polygon is defined by a series of contiguous subelements. Each subelement is a straight line or a circular arc. The first triplet is a header. The interpretation in the header denotes how many subelements (or additional triplets in `SDO_ELEM_INFO`) make up this compound polygon. All the subelements in a compound polygon must be of the element type 2 (line segments) with interpretation of 1 or 2, and all the subelements must be contiguous (that is, the last point of a subelement is the first point of the next subelement).

In the example in the slide, notice that the ordinate offsets of each subelement are different and correspond to where the subelement begins in the ordinate array.

**Note:** `SDO_GTYPE` for this geometry is 2003 (a single polygon element). Compound polygons have the same properties as polygons (that is, they cannot self-cross, and so on).

Instead of using an element type 5, it is recommended that you use the values 1005 or 2005 to clearly define whether it is an outer ring or an inner ring polygon. Outer ring compound polygons are represented as the element type 1005, and inner ring (void) compound polygons are represented as the element type 2005.

# Rules for Polygon Element Types

- Outer ring polygons (1003, 1005) must be followed by all their inner rings (2003, 2005).
- Outer ring polygons must be stored in the counterclockwise rotation.
- Inner ring polygons, also referred to as voids, must be stored in the clockwise rotation.

ORACLE





## Rules for Polygon Element Types

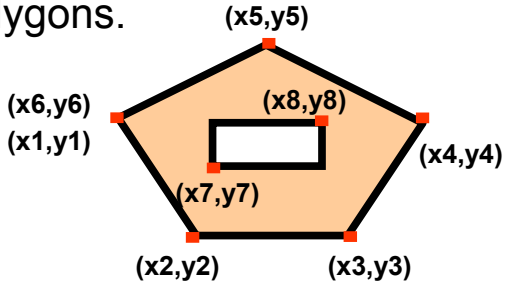
This slide summarizes the rules for representing polygons. Outer ring compound polygons are represented as the element type 1005, and inner ring (void) compound polygons are represented as the element type 2005.

Outer ring compound polygons must be stored with a counterclockwise rotation. Inner ring (void) compound polygons must be stored with a clockwise rotation. Enforcing the rotation enables faster area calculations.

An outer ring polygon must be followed by all its inner rings (voids). When you read geometries, this makes it easier to determine which rings are voids and which are not. Before this order enforcement, it was up to the application to mathematically determine which rings were outer rings and which were voids.

# Element: Polygon with a Void

- A void can be modeled with any combination of type 2003 and type 2005 elements.
- Voids can contain islands and islands can contain voids.
- The area is computed as the difference between the outer and inner polygons.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1003 | 1 |
| 13 | 2003 | 3 |

ORACLE



## Element: Polygon with a Void

This is an example of a polygon with a void. The outer ring has an element type of 1003, and the inner ring has an element type of 2003.

Voids can contain islands, and islands can contain voids. The area is computed as the difference between outer and inner polygons.

The rotation of outer ring elements must be counterclockwise, and the rotation of inner ring elements must be clockwise.

An outer ring element must be followed by all its inner ring elements before another outer ring element can be specified as part of the geometry.

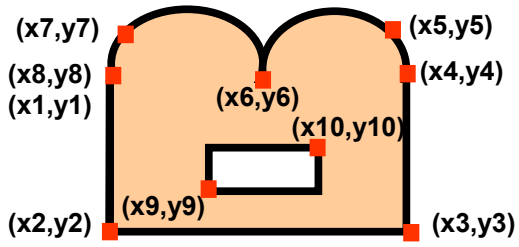
**Note:** `SDO_GTYPE` of this geometry is 2003 (a single 2D polygon).

# Element: Compound Polygon with a Void

Element type 1005: Compound polygon
Interpretation 2: Number of subelements

| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1005 | 2 |
| 1 | 2 | 1 |
| 7 | 2 | 2 |
| 17 | 2003 | 3 |

Element type 2003: Inner ring polygon
Interpretation 3: Optimized rectangle




ORACLE



## Element: Compound Polygon with a Void

This is an example of a compound polygon with a void. The outer ring has an element type of 1005 (compound polygon) and the inner ring has an element type of 2003 (optimized rectangle).

The compound polygon has three triplet value entries in the `SDO_ELEM_INFO` field. The first triplet is a header (1,1005,2). "2" denotes that two more triplet values follow to define the compound polygon subelements (1,2,1) and (7,2,2). Notice that the last triplet value for this geometry (17,2003,3) is not part of the compound polygon. It defines a void, which is a rectangle.

**Note:** `SDO_GTYPE` of this geometry is 2003 (a single 2D polygon). A polygon with one or more void elements can still be considered a single polygon (that is, not categorized as a multipolygon).

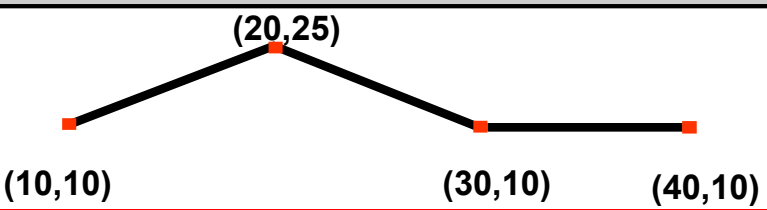
# Lesson Agenda

- MDSYS schema and spatial data management
- Spatial native data type: SDO_GEOMETRY
- Geometry elements
- Construction of geometries by using the INSERT statement
- Spatial metadata

# Constructing Geometries

- Use the SQL `INSERT` statement to construct geometries and insert them into a spatial layer.
- The `SDO_GEOMETRY` constructor is called to instantiate a geometry object.

```
INSERT INTO LINES VALUES (
attribute_1, …. attribute_n,
SDO_GEOMETRY (
  2002, null, null,
   SDO_ELEM_INFO_ARRAY (1,2,1),
      SDO_ORDINATE_ARRAY (
        10,10, 20,25, 30,10, 40,10))
);
```




ORACLE



## Constructing Geometries

Earlier, you saw a transactional insert for a point that is stored in the `SDO_POINT` field. This is the first transactional insert example for storing information in the array fields (`SDO_ELEM_INFO` and `SDO_ORDINATES`) of the `SDO_GEOMETRY` object.

The `INSERT` statement uses an `SDO_GEOMETRY` constructor, sets the `SRID` and `SDO_POINT` fields to null, and uses the `SDO_ELEM_INFO_ARRAY` and `SDO_ORDINATE_ARRAY` constructors to populate the array fields.

**Note:** When storing a `NULL` geometry column, the entire geometry column—and not each individual field in the geometry column—must be set to `NULL`. For example:

```
INSERT INTO LINES VALUES (attribute1, … attribute_n, NULL);
```

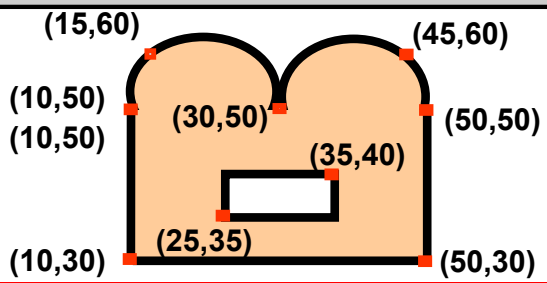
Do *not* set each field of the geometry object to `NULL`:

```
INSERT INTO LINES VALUES (attribute1, … attribute_n,
      SDO_GEOMETRY(NULL, NULL, NULL, NULL, NULL));
```

## Constructing Geometries

When you call a constructor, you use the type name and not the column name or field name.

```
INSERT INTO PARKS VALUES(
 attribute_1, …, attribute_n,
   SDO_GEOMETRY(
    2003, null, null,
    SDO_ELEM_INFO_ARRAY
    (1,1005,2, 1,2,1, 7,2,2, 17,2003,3),
     SDO_ORDINATE_ARRAY
      (10,50,10,30,50,30,50,50,45,60,
       30,50,15,60,10,50,25,35,35,40)));
```




ORACLE

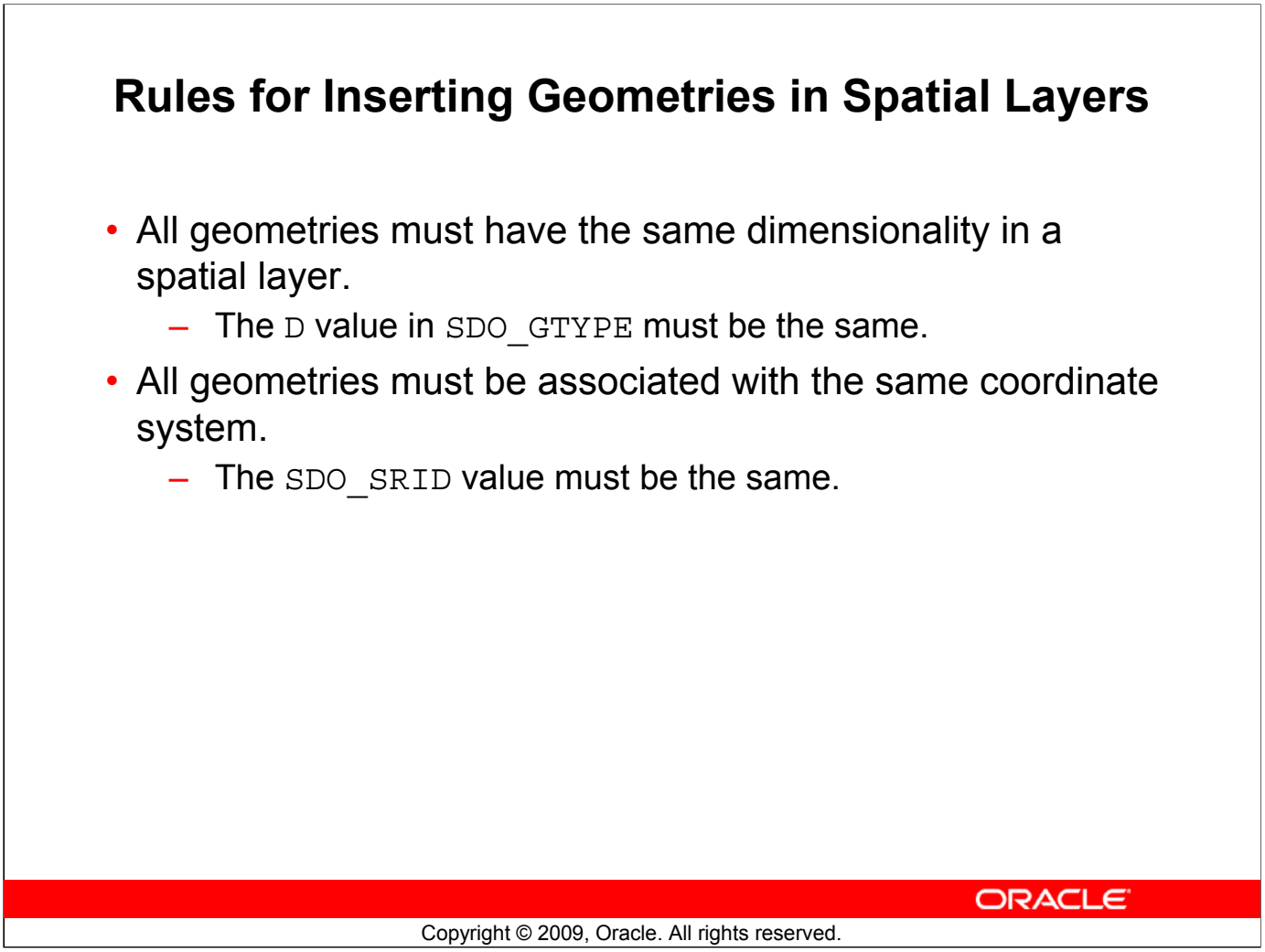


### Constructing Geometries (continued)

This is another transactional insert example for storing information in the array fields (SDO_ELEM_INFO and SDO_ORDINATES) of the SDO_GEOMETRY object. This example inserts the compound polygon with a void that was discussed earlier.

The INSERT statement uses an SDO_GEOMETRY constructor, sets the SRID and SDO_POINT fields to null, and uses SDO_ELEM_INFO_ARRAY and SDO_ORDINATE_ARRAY constructors to populate the array fields. Note that when you call a constructor, you use the type name and not the column name or the field name. For the SDO_ELEM_INFO field, the type name is SDO_ELEM_INFO_ARRAY and the SDO_ORDINATES field is of the SDO_ORDINATE_ARRAY type.

**Note:** SDO_GTYPE of this geometry is 2003 (a single 2D polygon). A polygon with one or more void elements can still be considered a single polygon (that is, not categorized as a multipolygon).




## Rules for Inserting Geometries in Spatial Layers

When you are inserting geometries in an SDO_GEOMETRY column, you must consider the following rules:

- In a spatial layer, you cannot have a mix of two-dimensional and three-dimensional geometries. All the geometries must have the same dimensionality in a spatial layer. That means that the D value of the SDO_GTYPE field must be the same.
- In a spatial layer, you cannot have geometries associated with different coordinate systems. All the geometries must be associated with the same coordinate system. That means that the SDO_SRID value must be the same for all the geometries in a single SDO_GEOMETRY column.

You may have a table with multiple SDO_GEOMETRY columns. In that case, each column may have a different dimensionality and coordinate system associated with it.

# Review: SDO_GTYPE Values and Geometry Types

| SDO_GTYPE | Four-digit GTYPEs: Include dimensionality | |
|---|---|---|
| | 2D | 3D |
| 1. POINT | 2001 | 3001 |
| 2. LINE STRING | 2002 | 3002 |
| 3. POLYGON | 2003 | 3003 |
| 4. HETEROGENEOUS COLLECTION | 2004 | 3004 |
| 5. MULTIPOINT | 2005 | 3005 |
| 6. MULTILINE STRING | 2006 | 3006 |
| 7. MULTIPOLYGON | 2007 | 3007 |

ORACLE

# Review: Element Types Summarized

| VALUE | Element Type | Interpretation Value |
|---|---|---|
| 1 | POINT | Number of points in the collection; or<br>0: Oriented point |
| 2 | LINE STRING | 1: Straight lines<br>2: Circular arcs |
| 3<br>1003<br>2003 | POLYGON<br>(Outer)<br>(Inner) | 1: Straight lines<br>2: Circular arcs<br>3: Optimized rectangle<br>4: Circle |
| 4 | COMPOUND LINE STRING | Number of type 2 subelements that make up the line string |
| 5 | COMPOUND POLYGON | Number of type 2 subelements that make up the polygon |
| 1005 | (Outer) | |
| 2005 | (Inner) | |

ORACLE

## Quiz: SDO_GTYPE and Element Type

- With this quiz, you should be able to explain the difference between SDO_GTYPE and element type.
- Determine SDO_GTYPE of each geometry described in the previous slides.

ORACLE



### Quiz: SDO_GTYPE and Element Type

After this exercise, you should be able to understand the difference between SDO_GTYPE and element type:

- SDO_GTYPE is the first field of the SDO_GEOMETRY object, and it describes the entire geometry (the big picture).
- A geometry may contain one or more elements. Each element's ordinates are stored in SDO_ORDINATE_ARRAY. Each element's type (or element type) is described as part of a triplet value stored in SDO_ELEM_INFO_ARRAY.

See the slide titled "Review: SDO_GTYPE Values and Geometry Types" (page 2-35) to determine the appropriate SDO_GTYPE of the geometries, as your instructor pages through the geometries covered in this lesson.

# Lesson Agenda

- MDSYS schema and spatial data management
- Spatial native data type: SDO_GEOMETRY
- Geometry elements
- Construction of geometries by using the INSERT statement
- Spatial metadata

# Spatial Metadata

- Update the USER_SDO_GEOM_METADATA view with metadata information about each SDO_GEOMETRY column.
- The following two views are available to each Spatial user:
  - USER_SDO_GEOM_METADATA
  - ALL_SDO_GEOM_METADATA:
    - This contains metadata information about all spatial tables that the user has SELECT privileges to.
    - Only select from this system view; never insert directly into this view.

ORACLE



## Spatial Metadata

For every SDO_GEOMETRY column, you must populate metadata into a dictionary view called USER_SDO_GEOM_METADATA. This view is created for you when you install Oracle Spatial. You are responsible for updating the USER_SDO_GEOM_METADATA view with metadata information about any SDO_GEOMETRY column that you create. Oracle Spatial ensures that the ALL_SDO_GEOM_METADATA view is updated to reflect the rows that you insert into USER_SDO_GEOM_METADATA. The ALL_SDO_GEOM_METADATA view contains metadata information about all the spatial tables that you have SELECT privileges to. This view contains all the metadata for the Oracle user that you are logged in as, and all metadata for geometry columns that the user has SELECT access to.

The information in this view is used by Oracle Spatial and by third-party tools and applications. Oracle Spatial never updates or deletes information in the USER_SDO_GEOM_METADATA view.

**Note:** You must update this view before creating spatial indexes on your spatial layers.

## USER_SDO_GEOM_METADATA

```
DESCRIBE USER_SDO_GEOM_METADATA
 Name            Null?       Type
 --------------- ----------  ------------------
 TABLE_NAME      NOT NULL    VARCHAR2(32)
 COLUMN_NAME     NOT NULL    VARCHAR2(1024)
 DIMINFO                     SDO_DIM_ARRAY
 SRID                        NUMBER
```

- MDSYS.SDO_DIM_ARRAY:

```
VARRAY(4) OF SDO_DIM_ELEMENT
```

- The MDSYS.SDO_DIM_ELEMENT object:

```
 SDO_DIMNAME     VARCHAR2(64)
 SDO_LB          NUMBER
 SDO_UB          NUMBER
 SDO_TOLERANCE   NUMBER
```

ORACLE



### USER_SDO_GEOM_METADATA

Here you see what the USER_SDO_GEOM_METADATA dictionary view looks like.

**TABLE_NAME:** The name of the table that contains the SDO_GEOMETRY column

**COLUMN_NAME:** The SDO_GEOMETRY column name

**DIMINFO:** VARRAY of SDO_DIM_ELEMENT objects. This is where you define the axis for your coordinate system (the 2, 3, or 4 axis can be defined). The fields of the SDO_DIM_ELEMENT object are described in the next slide.

**SRID:** SRID is an acronym for the spatial reference system ID. This must either be NULL or match a value in the SRID column of the MDSYS.CS_SRS table, where all the supported coordinate systems are defined.

## USER_SDO_GEOM_METADATA: SDO_DIM_ELEMENT Object

- SDO_DIMNAME:
  - The dimension name
- SDO_LB:
  - The lower bound for this dimension
- SDO_UB:
  - The upper bound for this dimension

ORACLE



### USER_SDO_GEOM_METADATA: SDO_DIM_ELEMENT Object

SDO_DIMNAME, SDO_LB, and SDO_UB, SDO_TOLERANCE are fields of the SDO_DIM_ELEMENT object.

- **SDO_DIMNAME:** A character string to identify the axis. This value is not currently used by Oracle Spatial.
- **SDO_LB:** LB is an acronym for lower bound. It represents the lowest possible value for this dimension.
- **SDO_UB:** UB is an acronym for upper bound. It represents the largest possible value for this dimension.

R-tree indexes are flexible and can accommodate geometries that fall outside the SDO_LB-to-SDO_UB range. Spatial validation routines flag data that fall outside the coordinate system bounds as invalid. For geodetic data, the bounds of the coordinate system must always be set to –180,180 and –90,90.

## USER_SDO_GEOM_METADATA: SDO_DIM_ELEMENT Object

SDO_TOLERANCE:

- Is the distance required between two coordinates so that they are considered as individual coordinates
- Is used by Oracle Spatial indexing, operators, and functions:
  - Projected data: The tolerance unit and data unit are the same. Match decimal precision of ordinates with zeros, and append a 5 (for example, 0.005).
  - Geodetic data: For data stored in longitude or latitude, the tolerance unit is meters (that is, 0.5 is a one-half meter tolerance). The smallest tolerance allowed for geodetic data is 0.05 meters (5 centimeters).

ORACLE



### USER_SDO_GEOM_METADATA: SDO_DIM_ELEMENT Object (continued)

**SDO_TOLERANCE:** The distance required between two coordinates so that they are not considered the same coordinate. For projected data (that is, not longitude or latitude data) or data for which the SRID has not been set, this value is specified in the same unit as the data. Match the decimal precision of ordinates with zeros and append a 5 (that is, 0.005).

For geodetic data (that is, data represented in longitude or latitude), this value is specified in meters (that is, 0.05 is a 5-centimeter tolerance).

**Note:** For geodetic data, the smallest tolerance allowed is 0.05 meters. Oracle Spatial uses a tolerance of 0.05 meters even if a smaller tolerance is specified.

# Populating the USER_SDO_GEOM_METADATA View

```
INSERT INTO USER_SDO_GEOM_METADATA
(TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
VALUES (
'ROADS',
'GEOMETRY',
SDO_DIM_ARRAY (
      SDO_DIM_ELEMENT('Long', -180, 180, 0.5),
      SDO_DIM_ELEMENT('Lat',   -90,  90, 0.5)),
8307);
```

- For geodetic data, x-axis bounds *must* be –180 to 180, and y-axis bounds must be –90 to 90.
- Set the `SRID` column of the `USER_SDO_GEOM_METADATA` view and the `SDO_SRID` field of each geometry in a layer to the same spatial reference ID or to `NULL`.

ORACLE



## Populating the USER_SDO_GEOM_METADATA View

For every geometry column, one row of metadata must be inserted into `USER_SDO_GEOM_METADATA`. This is done with an ordinary transactional insert.

**Note:** You must specify the x-axis before the y-axis.

# Summary

In this lesson, you should have learned how to:

- Describe the schema associated with Oracle Spatial
- Create a table with a spatial layer
- Construct geometries of different element types
- Update the dictionary view with metadata information about spatial layers

ORACLE



## Summary

In this lesson, you learned about the `SDO_GEOMETRY` native data type in more detail. You also learned how to create a table with a spatial layer, insert different types of geometries, and update the spatial metadata.

# Practice 2: Overview

This practice covers the following topics:

- Creating tables with the `SDO_GEOMETRY` column
- Inserting geometrical elements of different types
- Updating the `USER_SDO_GEOM_METADATA` view

ORACLE



## Practice 2: Overview

In Part 1 of this practice, you create a table, `geom_data`, with a spatial column and insert different types of geometries in the table.

In Part 2 of this practice, you create some more tables for storing spatial data. The data in these tables is loaded later in Practice 5-1. You also update the `USER_SDO_GEOM_METADATA` view.

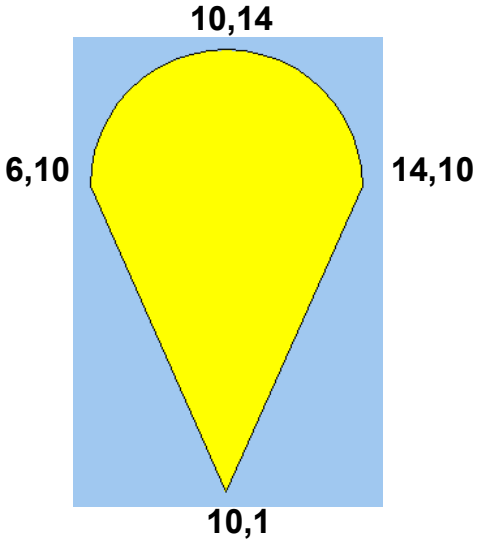
## Practice 2

### Part 1

1. Start SQL*Plus and log in using `student/student` as the username/password.
2. Describe the `SDO_GEOMETRY` data type and review its attributes and member functions.
3. Create a table, `geom_data`, with the following structure:
   i. `Object_id number PRIMARY KEY`
   ii. `Name varchar2(30)`
   iii. `Shape SDO_GEOMETRY`
4. Insert a two-dimensional point with the following coordinates and nonspatial attribute values. Use the `SDO_POINT` field. Set `SRID` to `NULL`.
   i. Object_id: `1`
   ii. Name: `Point`
   iii. Coordinates: `X=12`, `Y=14`
5. Insert a line string composed of straight lines by using the following values:
   i. Object_id: `2`
   ii. Name: `Line String`
   iii. Coordinates: `(10,25)`, `(20,30)`, `(25,25)`, `(30,30)`
   iv. Set `SRID` to `NULL`.
6. Insert an optimized rectangle by using the following values:
   i. Object_id: `3`
   ii. Name: `Rectangle`
   iii. Coordinates: `(1,1)`, `(5,7)`
   iv. Set `SRID` to `NULL`.
7. Insert a polygon with a hole by using the following values:
   i. Object_id: `4`
   ii. Name: `Polygon with a hole`
   iii. Coordinates:
   1.Outer ring polygon: `(2,4)`, `(4,3)`, `(10,3)`, `(13,5)`, `(13,9)`, `(11,13)`, `(5,13)`, `(2,11)`, `(2,4)`
   2.Inner ring polygon: `(7,5)`, `(7,10)`, `(10,10)`, `(10,5)`, `(7,5)`

If you want an extra challenge, complete the following exercise:

8. Insert a compound polygon with the following values:
   i. Object_id: `5`
   ii. Name: `Compound Polygon`
   iii. Shape:



## Practice 2 (continued)

### Part 2

9. Create the following tables:
   a. GEOD_CITIES

| Column Name | Data Type | Description |
|---|---|---|
| LOCATION | SDO_GEOMETRY | City location |
| CITY | VARCHAR2(42) | City name |
| STATE_ABRV | VARCHAR2(2) | State code |
| POP90 | NUMBER | Population (1990) |
| RANK90 | NUMBER | Population ranking (1990) |

   b. GEOD_INTERSTATES

| Column Name | Data Type | Description |
|---|---|---|
| HIGHWAY | VARCHAR2(35) | Interstate name |
| GEOM | SDO_GEOMETRY | Interstate geometry |

10. Describe the structure of the USER_SDO_GEOM_METADATA view.
11. Insert metadata for the following layers into the USER_SDO_GEOM_METADATA view:
    i. geom_data (SHAPE):
       i. Set SRID to NULL.
       ii. Set tolerance to 0.005.
       iii. For SDO_DIM_ARRAY, set the x-axis in the range from 0 to 300, and y-axis in the range from 0 to 300.
    ii. GEOD_CITIES (LOCATION):
       i. Set SRID to 8307.
       ii. Set tolerance to 0.05.
       iii. For SDO_DIM_ARRAY, set the Long axis in the range from –180.0 to 180.0 and Lat axis in the range from –90.0 to 90.0.
    iii. GEOD_INTERSTATES (GEOM):
       i. Set SRID to 8307.
       ii. Set tolerance to 0.05.
       iii. For SDO_DIM_ARRAY, set the Long axis in the range from –180.0 to 180.0 and Lat axis in the range from –90.0 to 90.0.

**Note:** Spatial data is loaded into the GEOD_CITIES and GEOD_INTERSTATES tables in the lesson titled, "Loading Spatial Data." Make sure you commit the inserted rows.

# 3

# Defining Collection Geometries

ORACLE

# Objectives

After completing this lesson, you should be able to do the following:

- Construct collection geometries such as multipoint, multiline, and multipolygon
- Describe how to construct an oriented point
- Identify `SDO_GEOMETRY` constructors and member methods

ORACLE



## Objectives

In this lesson, you learn how to construct collection geometries such as a multipoint, multiline, and multipolygon. You also learn to describe how to construct an oriented point. Additionally, the lesson gives more information about the `SDO_GEOMETRY` member methods and constructors.

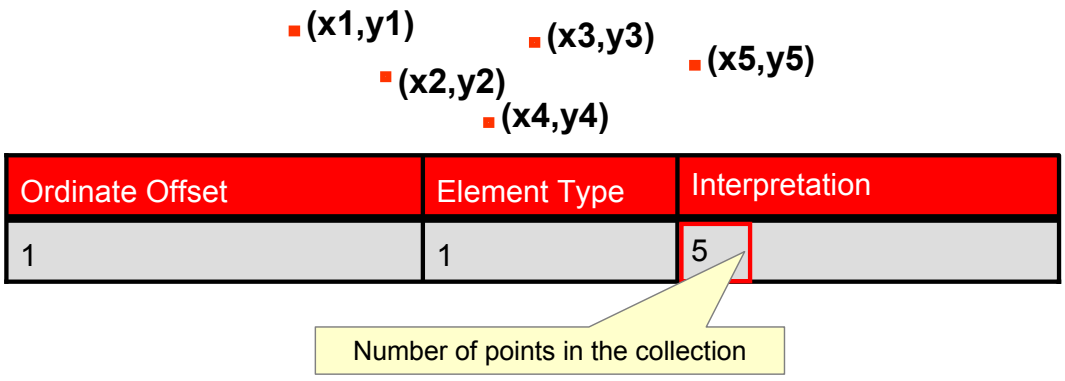
# Lesson Agenda

- Collection geometries:
  - Multipoint
  - Multiline string
  - Multipolygon
- Oriented point
- `SDO_GEOMETRY` constructors and member methods

ORACLE

# Collection: Multipoint

- A collection is a geometry that contains multiple elements.
- A geometry contains multiple points.
- SDO_GTYPE is 2005, which is a two-dimensional multipoint.






## Collection: Multipoint

This is the first example of collection geometry. A collection is a geometry that contains multiple elements. In this slide, the collection is a multipoint. For a multipoint, the element type is 1, and the interpretation corresponds to the number of points in the collection.

**Note:** SDO_GTYPE for this geometry is 2005 (2D multipoint). Remember that SDO_GTYPE is the first field of the SDO_GEOMETRY object and that it describes the entire geometry (the big picture).

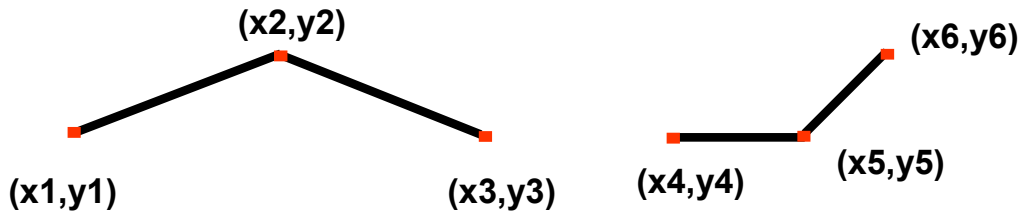
# Multipoint: Example

- The SDO_GTYPE 2005 indicates a two-dimensional multipoint.
- The interpretation value in the first triplet of SDO_ELEM_INFO_ARRAY defines the number of points.

```
SDO_GEOMETRY (2005, NULL, NULL,
  SDO_ELEM_INFO_ARRAY (1,1,2),
  SDO_ORDINATE_ARRAY (65,5, 70,7))
```

# Collection: Multiline String

- A geometry contains multiple nonconnected line string, circular arc string, or compound line string elements.
- Example: A street intersected by a park




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 2 | 1 |
| 7 | 2 | 1 |

ORACLE



## Collection: Multiline String

In this slide, the collection is a multiline string. For a multiline string, each element in the collection contains a triplet value in the `SDO_ELEM_INFO` field. In the example in the slide, the straight line string triplet is (1,2,1) and the second line string triplet is (7,2,1).

The boundaries of a multiline string geometry are the end points of each line string element, unless the end points overlap. Then, only the end points that overlap an odd number of times are boundaries.

In the example in the slide, the boundaries are x1,y1, x3,y3, x4,y4, and x6,y6.

**Note:** `SDO_GTYPE` for this geometry is 2006 (2D multiline string). Remember that `SDO_GTYPE` is the first field of the `SDO_GEOMETRY` object and that it describes the entire geometry (the big picture).

# Multiline String: Example

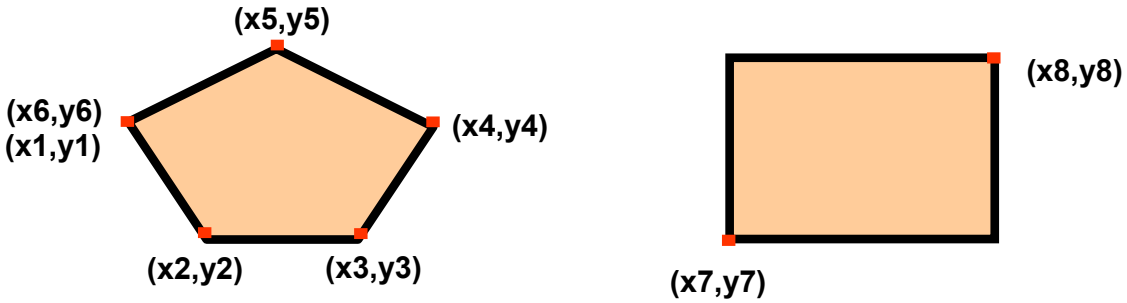
The SDO_GTYPE 2006 indicates a two-dimensional multiline.

```
SDO_GEOMETRY (2006, NULL, NULL,
  SDO_ELEM_INFO_ARRAY (1,2,1, 7,2,1),
  SDO_ORDINATE_ARRAY (0,0, 1,1, 2,0,
                      4,0, 5,0, 6,6))
```

ORACLE

# Collection: Multipolygon

A geometry contains multiple exterior polygons.




| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1003 | 1 |
| 13 | 1003 | 3 |

ORACLE



## Collection: Multipolygon

In this slide, the collection is a multipolygon. For a multipolygon, each element in the collection contains a triplet value in the `SDO_ELEM_INFO` field. In the example in the slide, the pentagon polygon element triplet is (1,1003,1) and the optimized rectangle triplet is (13,1003,3).

The state of Hawaii can be represented as a multipolygon.

**Note:** `SDO_GTYPE` for this geometry is 2007 (2D multipolygon).

# Multipolygon: Example

- The SDO_GTYPE 2007 indicates a two-dimensional multipolygon.
- The element type 1003 indicates an outer ring polygon.

```
SDO_GEOMETRY (2007, NULL, NULL,
  SDO_ELEM_INFO_ARRAY (1,1003,1, 13,1003,3),
  SDO_ORDINATE_ARRAY (1,1, 2,0, 4,0, 5,1,
                      3,2, 1,1, 10,0, 15,1))
```

ORACLE

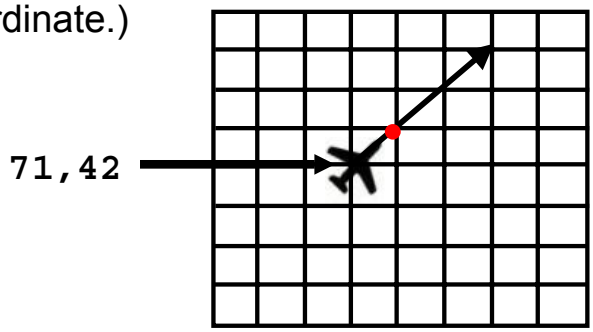
# Lesson Agenda

- Collection geometries:
  - Multipoint
  - Multiline string
  - Multipolygon
- Oriented point
- `SDO_GEOMETRY` constructors and member methods

ORACLE

# Oriented Point

- Is used for orienting symbology or text for maps and other display applications
- Includes:
  - The coordinates of the point (the location of the airplane in the following image)
  - An additional coordinate that defines the direction of the orientation vector (The vector starts at the location of the airplane and continues in the direction of the additional coordinate.)




ORACLE



## Oriented Point

An oriented point support is useful for orienting symbology that is used in mapping and other display applications. An oriented point can be used in map visualization and display applications that include symbols, such as a shield symbol to indicate a highway.

Oriented points contain coordinates for the point being stored, as well as an additional coordinate that defines the direction of the orientation vector. The origin (0,0) of the orientation vector is located on the coordinate being stored (71,42) in the example in the slide. An additional coordinate defines the direction of the orientation vector. The vector starts at the coordinate being stored, and continues in the direction of the additional coordinate. The orientation vector is used to orient symbology and/or text for labeling in mapping and other display applications. For example, you can use an orientation vector to show a text label or an image oriented at an angle beside a point.

## SDO_ELEM_INFO Triplet Values

- The SDO_ELEM_INFO field contains two triplets.
- The first triplet defines the point.
- The second triplet with interpretation set to 0 indicates that there is an orientation vector.

| Ordinate Offset | Element Type | Interpretation |
|---|---|---|
| 1 | 1 | 1 |
| 3 | 1 | 0 |

ORACLE



### SDO_ELEM_INFO Triplet Values

The SDO_ELEM_INFO field for an oriented point contains two triplets. The first triplet has:

- **Ordinate offset:** The element offset of the point
- **Element type:** An element type of 1 to denote a point element type
- **Interpretation:** An interpretation of 1 to denote a single point

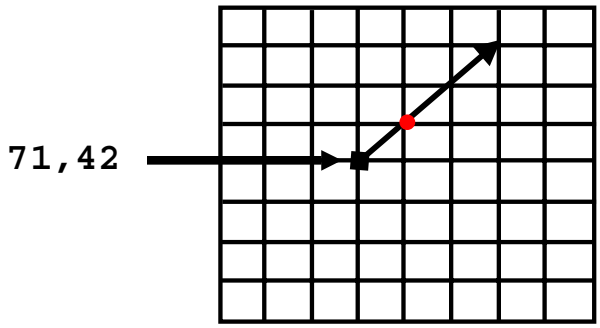
The second triplet has:

- **Ordinate offset:** The element offset for the orientation vector
- **Element type:** An element type of 1 to denote a point element type
- **Interpretation:** An interpretation of 0 to specify that this is an orientation vector for the point

# Oriented Point: Example

- The point location is `(71, 42)`.
- A symbol associated with the point would be oriented along a vector from `(71, 42)` through `(72, 43)`, at a 45-degree angle.

```
SDO_GEOMETRY(2001, NULL, NULL,
  SDO_ELEM_INFO_ARRAY(1,1,1, 3,1,0),
  SDO_ORDINATE_ARRAY(71, 42,
                     1, 1))
```




ORACLE



## Oriented Point: Example

Oriented points contain coordinates for the point being stored, as well as a set of coordinates that are relative to the point specified. Given the coordinates of the orientation vector (i, j), the following equation can be used to calculate the orientation vector angle:

```
Angle = arctan (j/i)
```

In the orientation vector example in the slide, arctan(1/1) = 45 degrees.

# Rules for an Oriented Point

- The point must be followed by its orientation.
- The origin (0,0) of the orientation vector is located at the coordinates of the point that it is associated with.
- Multipoint oriented points are allowed, but orientation information must follow the point being oriented.
- Orientation vector values must be between –1 and 1.

ORACLE



## Rules for an Oriented Point

The slide lists the rules that you should know when defining an oriented point. The following list contains some more rules:

- A 2D point has a 2D orientation vector.
- A 2D point with a Linear Referencing System (LRS) measure has a 2D orientation vector.
- A 3D point has a 3D orientation vector.
- A 3D point with an LRS measure has a 3D orientation vector.

# Review: SDO_GTYPE Values and Geometry Types

| SDO_GTYPE | Four-digit GTYPEs: Include dimensionality | |
|---|---|---|
| | 2D | 3D |
| 1. POINT | 2001 | 3001 |
| 2. LINE STRING | 2002 | 3002 |
| 3. POLYGON | 2003 | 3003 |
| 4. HETEROGENEOUS COLLECTION | 2004 | 3004 |
| 5. MULTIPOINT | 2005 | 3005 |
| 6. MULTILINE STRING | 2006 | 3006 |
| 7. MULTIPOLYGON | 2007 | 3007 |

ORACLE

# Review: Element Types Summarized

| VALUE | Element Type | Interpretation Value |
|---|---|---|
| 1 | POINT | Number of points in the collection; or<br>0: Oriented point |
| 2 | LINE STRING | 1: Straight lines<br>2: Circular arcs |
| 3<br>1003<br>2003 | POLYGON<br>(Outer)<br>(Inner) | 1: Straight lines<br>2: Circular arcs<br>3: Optimized rectangle<br>4: Circle |
| 4 | COMPOUND LINE STRING | Number of type 2 subelements that make up the line string |
| 5 | COMPOUND POLYGON | Number of type 2 subelements that make up the polygon |
| 1005<br>2005 | (Outer)<br>(Inner) | |

ORACLE

# Quiz: SDO_GTYPE and Element Type

- This quiz should help you understand the difference between SDO_GTYPE and element type.
- Determine SDO_GTYPE of each geometry described in the previous slides.

ORACLE



## Quiz: SDO_GTYPE and Element Type

At the end of this exercise, you should be able to understand the difference between SDO_GTYPE and element type:

- SDO_GTYPE is the first field of the SDO_GEOMETRY object, and it describes the entire geometry (the big picture).
- A geometry may contain one or more elements. Each element's ordinates are stored in SDO_ORDINATE_ARRAY. Each element's type (or element type) is described as part of a triplet value stored in SDO_ELEM_INFO_ARRAY.

See the slide titled "Review: SDO_GTYPE Values and Geometry Types" (page 3-15) to determine the appropriate SDO_GTYPE of the geometries, as your instructor pages through the geometries covered in this lesson.

# Lesson Agenda

- Collection geometries:
  - Multipoint
  - Multiline string
  - Multipolygon
- Oriented point
- `SDO_GEOMETRY` constructors and member methods

## SDO_GEOMETRY Constructors

- SDO_GEOMETRY objects can be created using different overloaded constructors.
- The constructor input can be either of the following:
  - SQL Multimedia (SQL/MM) Well-Known Text (WKT) string in the character large object (CLOB) or VARCHAR2 format
  - SQL/MM Well-Known Binary (WKB) object in the binary large object (BLOB) format
- Constructors return an SDO_GEOMETRY object.

ORACLE



### SDO_GEOMETRY Constructors

The SDO_GEOMETRY object type has constructors that create a geometry object from a Well-Known Text (WKT) string in the CLOB or VARCHAR2 format, or from a Well-Known Binary (WKB) object in the BLOB format.

# Constructors for SDO_GEOMETRY

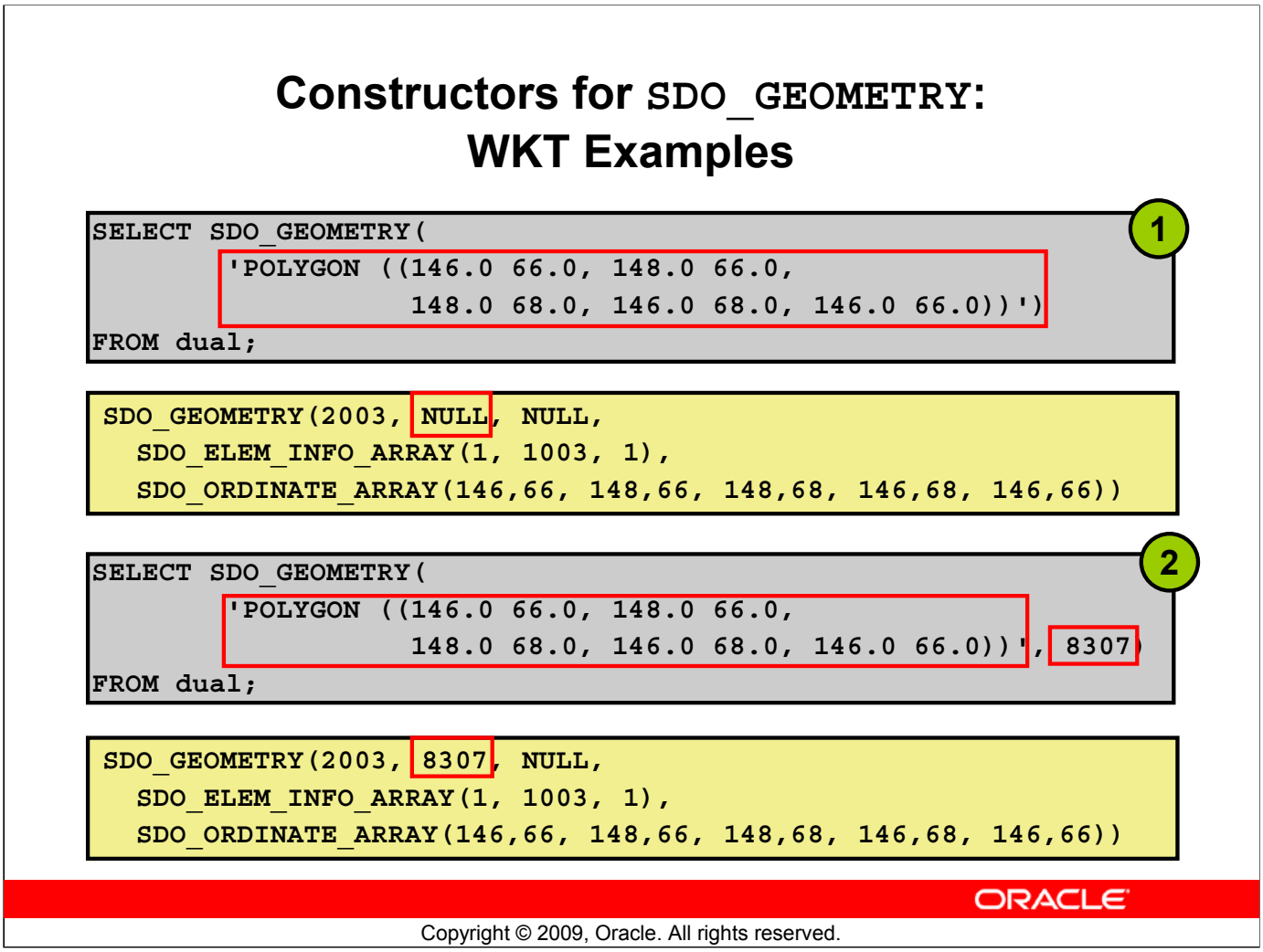
Constructor formats:

```
SDO_GEOMETRY(wkt CLOB, srid NUMBER DEFAULT NULL);
```

```
SDO_GEOMETRY(wkt VARCHAR2, srid NUMBER DEFAULT NULL);
```

```
SDO_GEOMETRY(wkb BLOB, srid NUMBER DEFAULT NULL);
```

ORACLE





## Constructors for SDO_GEOMETRY: WKT Examples

These examples show the use of SDO_GEOMETRY constructors that take SQL/MM WKT as input and change it to SDO_GEOMETRY.

1. This example takes a WKT string as input. In a WKT, spaces separate ordinates of a vertex, and commas separate vertices.
2. This example includes a spatial reference system ID to populate the SDO_SRID value of the SDO_GEOMETRY object.

# SDO_GEOMETRY Member Methods

- The SDO_GEOMETRY object type has member methods that retrieve information about a geometry object.
- Tables must be aliased when accessing attributes or member methods on object columns.

## GET_GTYPE() Method

- The GET_GTYPE() method returns the geometry type of a geometry object, as specified in its SDO_GTYPE value.

```
SELECT g.geom.get_gtype() state_gtype
FROM   geod_states g
WHERE  g.state = 'Delaware';
```

```
STATE_GTYPE
-----------
          3
```

- In the query, the GEOD_STATES table is aliased. This enables access to the GET_GTYPE() method of the SDO_GEOMETRY object.



### GET_GTYPE() Method

GET_GTYPE() returns the geometry type of the SDO_GEOMETRY object.

**Note:** Whenever a method or an attribute associated with an object is accessed, a table alias must be used. In the example in the slide, the SDO_GEOMETRY object method GET_GTYPE is used. The table alias used in this example is g. geom is the name of the SDO_GEOMETRY column as defined in the geod_states table.

## GET_DIMS() Method

Returns the number of dimensions of a geometry object, as specified in its SDO_GTYPE value

```
SELECT s.geom.get_dims() state_dims
FROM geod_states s
WHERE s.state = 'Delaware';
```

```
STATE_DIMS
----------
         2
```

ORACLE



### GET_DIMS() Method

The GET_DIMS() method returns the dimensionality of the geometry.

As already mentioned, a method or an attribute associated with an object that is accessed must use a table alias. Therefore, the table alias used is s.

**Note:** All geometries in the layer must have the same dimensionality and LRS dimension. Applications must either call this function only once per layer in a separate statement (not in the context of an operator) or parse SDO_GTYPE.

## Some More Member Methods for SDO_GEOMETRY

- GET_WKB():
  - Returns the WKB format of the geometry
- GET_WKT():
  - Returns the WKT format of the geometry
- ST_CoordDim():
  - Returns the dimensionality of the geometry (same as the GET_DIMS() method)
- ST_IsValid():
  - Validates the geometry
  - Returns 1 if valid and 0 if not valid

ORACLE



### Some More Member Methods for SDO_GEOMETRY

There are several SQL/MM-compliant member methods for the SDO_GEOMETRY object. These member methods are:

- **GET_WKB():** Returns the Well-Known Binary format of the geometry as defined by SQL/MM
- **GET_WKT():** Returns the Well-Known Text format of the geometry as defined by SQL/MM
- **ST_CoordDim()**: Returns the dimensionality of the geometry
- **ST_IsValid():**
  - Validates the geometry
  - Returns 1 if valid, and 0 if invalid
  - Always uses 0.001 tolerance

  **Note:** Usually, you use SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT to determine whether the data is truly invalid (at true tolerance) and to find out more information about why the data is invalid. Some validation and debugging functions are discussed in detail in the lesson titled "Validating and Debugging Geometries."

# Using the SDO_GEOMETRY Member Methods: Example

```
set long 500;
SELECT a.geom.GET_WKT()
FROM (SELECT SDO_GEOMETRY(2003, 8307, NULL,
                SDO_ELEM_INFO_ARRAY(1, 1003, 1),
                SDO_ORDINATE_ARRAY(146,66, 148,66,
                                   148,68, 146,68,
                                   146,66)) geom
      FROM dual) a;
```

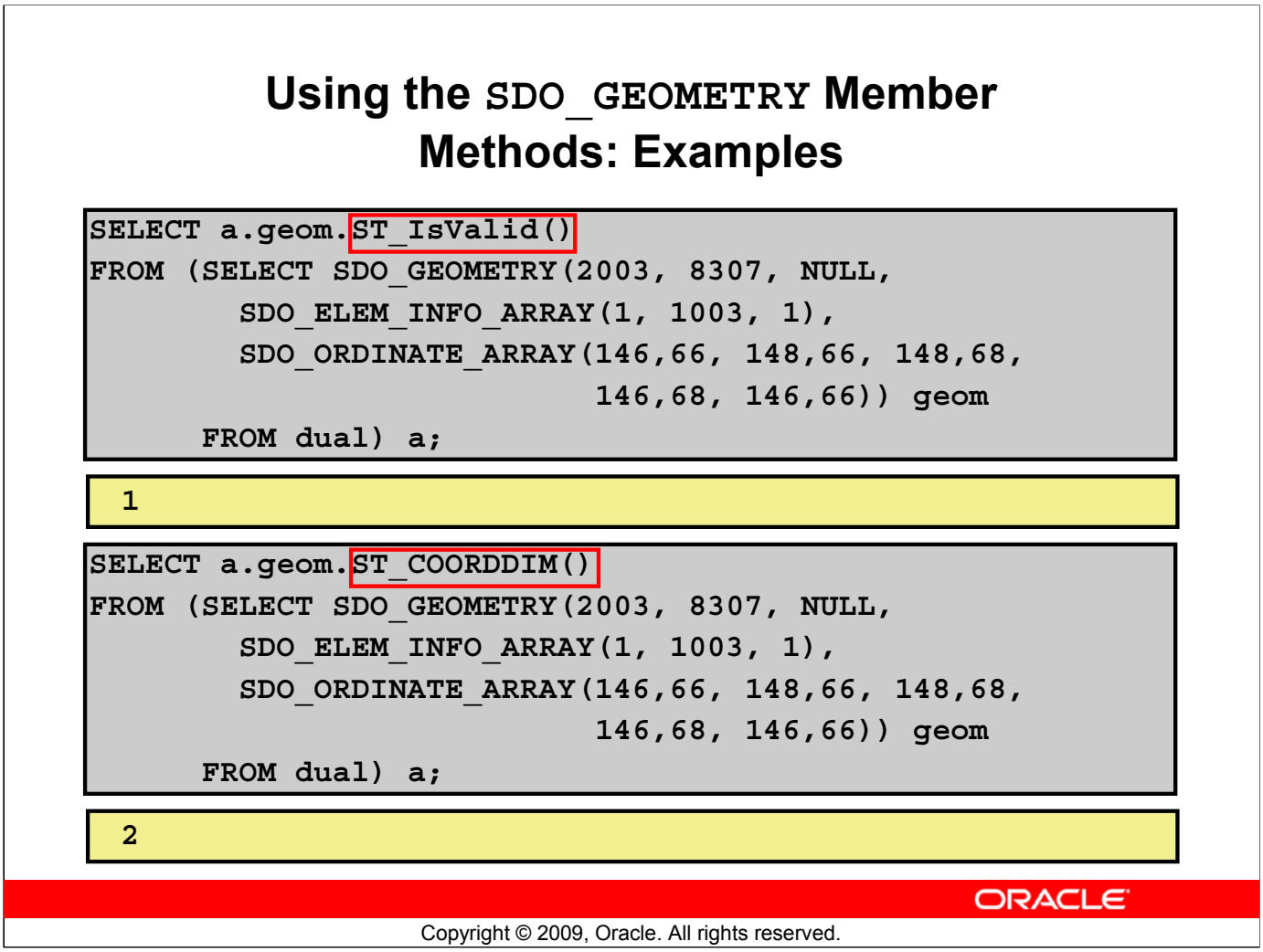
```
A.GEOM.GET_WKT()
----------------
POLYGON ((146.0 66.0, 148.0 66.0, 148.0 68.0, 146.0
          68.0, 146.0 66.0))
```



## Using the SDO_GEOMETRY Member Methods: Example

This example shows the use of the SDO_GEOMETRY member method GET_WKT().

Note that the returned SQL/MM WKT does not include coordinate system information.




## Using the SDO_GEOMETRY Member Methods: Examples (continued)

These examples show the use of the SDO_GEOMETRY member methods: ST_IsValid() and ST_CoordDim(). Note that the ST_IsValid() function returns 1 or 0. For more comprehensive reporting, use SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT.

# Summary

In this lesson, you should have learned how to:

- Construct collection geometries such as multipoints, multilines, and multipolygons
- Define an oriented point
- Use `SDO_GEOMETRY` constructors and member methods

ORACLE



## Summary

In this lesson, you learned about collection geometries and the oriented point geometry. This lesson also introduced the `SDO_GEOMETRY` constructors and member methods.

# Practice 3: Overview

This practice covers the following topics:

- Inserting collection geometries
- Using the `SDO_GEOMETRY` constructors and methods

ORACLE



## Practice 3: Overview

In this practice, you insert collection geometries such as multipoint, multiline, and multipolygon. You also run some `SDO_GEOMETRY` constructors and methods.

## Practice 3

1. Insert an oriented point in the `geom_data` table by using the following values:
   i. Object_id: `6`
   ii. Name: `Oriented Point`
   iii. Coordinates: `(12,14)`
   iv. Orientation vector values: `(0.3, 0.2)`
2. Insert a multiline geometry composed of two line strings. Use the following values:
   i. Object_id: `7`
   ii. Name: `Multiline`
   iii. Coordinates: `(50,22, 60,22) (65,20, 65,25)`
3. Insert a multipolygon geometry composed of two rectangles touching at one point. Use the following values:
   i. Object_id: `8`
   ii. Name: `Multipolygon-touching`
   iii. Coordinates: `(50,115, 65,125, 65,125, 75,130)`
4. Write a query to get the dimension and `GTYPE` of the geometry object with Object_id as `4`.
5. Write a query to get the Well-Known Text format of the geometry object with Object_id as `5`.
   **Note:** At SQL prompt, run `set long 150` to see the full output.

If you want extra challenge, complete the following exercise:

6. Insert a point geometry by using a Well-Known Text string constructor into the `geom_data` table. Set Object_id to `9` and Name to `point`. The point coordinates are (80, 40).

# 4

# Associating Spatial Layers with Coordinate Systems

ORACLE

# Objectives

After completing this lesson, you should be able to do the following:

- Define coordinate systems
- Explain the difference between georeferenced and nongeoreferenced coordinate systems
- Describe the concepts of geodetic coordinate systems
- Define the whole Earth geometry model
- Describe the coordinate system transformation functions
- Identify units supported by Oracle Spatial

ORACLE



## Objectives

This lesson provides an overview of the concepts associated with coordinate systems.

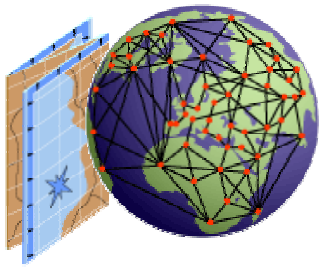
# Lesson Agenda

- Coordinate systems:
  - Types of coordinate systems:
    - Projected
    - Geodetic
- Geodetic coordinate system concepts:
  - Geodetic optimized rectangles
  - `CS_SRS` dictionary view
  - Restrictions in geodetic coordinate systems
- Whole earth geometry model and tolerance
- Coordinate system transformations
- Units supported by Oracle Spatial

ORACLE

# Coordinate System

- Defines a means of assigning coordinates to a location
- Establishes the relationship between sets of coordinates
- Enables interpretation of a set of coordinates as a location in the real-world space



ORACLE



## Coordinate System

Coordinate systems are used to identify where an object is in space. This location of an object is considered absolute with respect to the coordinate system used.

Coordinate systems are also used to identify where an object is in relation to other objects. The location of objects is relative to each other within the specified coordinate system.

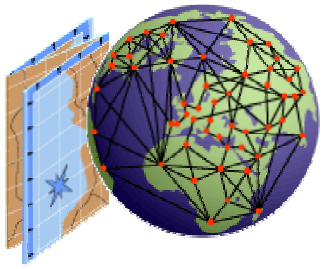
Any spatial data is defined using coordinates based on a standard coordinate system. A coordinate system is also referred to as a *spatial reference system*. Spatial data always has a coordinate system associated with it.

So far in this course, you did not specify the coordinate system (that is, the spatial reference ID [`SRID`] was `NULL` in `USER_SDO_GEOM_METADATA` and the `SDO_SRID` field of the geometry object was `NULL`). In this lesson, you see how you can define the coordinate system explicitly by setting the `SRID` column in the `USER_SDO_GEOM_METADATA` and `SDO_SRID` fields for each geometry in the layer.

# Types of Coordinate Systems

There are two types of coordinate systems:

- Georeferenced: Related to a specific representation of the Earth
  - Geodetic coordinate systems use longitude/latitude.
  - Projected coordinate systems do not use longitude/latitude. They use planar Cartesian coordinates to represent a location on the Earth.
- Nongeoreferenced: Not related to any representation of the Earth
  - Local coordinate systems use Cartesian coordinates to represent non-Earth-related data.



ORACLE



## Types of Coordinate Systems

There are two kinds of coordinate systems. Georeferenced coordinate systems are Earth related. Local or nongeoreferenced coordinate systems are not Earth related. Oracle Spatial supports the following types of coordinate systems:

- **Geodetic:** A georeferenced coordinate system that uses angular coordinates such as latitude and longitude. Longitude specifies positions east and west of the prime meridian, and latitude specifies positions north and south of the equator. In Oracle Spatial, geodetic data is expressed in decimal degrees varying from –180 degrees to 180 degrees in longitude, and from –90 degrees to 90 degrees in latitude. Lines of latitude are parallel to each other. The lengths of parallels decrease as the parallels move away from the equator until their lengths equal 0 at the North Pole and South Pole. Lines of longitude are not parallel, but converge at the poles. The distance or length specified by a degree of longitude changes from about 111 kilometers at the equator to zero at the poles.
- **Projected:** A georeferenced coordinate system that uses planar Cartesian coordinates determined by performing a mathematical mapping from a point on the Earth's surface to a plane. Projected information is specified in units of length such as meters or feet.
- **Local:** Cartesian coordinates in a nongeoreferenced coordinate system used for computer-aided design (CAD) applications and local surveys. Generically, a Cartesian coordinate is derived by measuring the position of a point from a defined origin along axes that are perpendicular in the represented two-dimensional or three-dimensional space.

## Projected Coordinate System

- Any two points are connected with a straight line.
  - The shortest path between two points is through that line.
- There is only one way to connect two points.
- A polygon can define only one closed area.

ORACLE



### Projected Coordinate System

Because projected data is Cartesian in nature, it has certain characteristics that differentiate it from geodetic data:

- Any two points are connected with a straight line that represents the shortest distance between those points.
- There is only one possible line that can represent the shortest distance between two points.
- A polygon defines only one closed area.

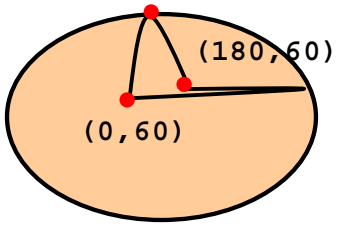
# Lesson Agenda

- Coordinate systems:
  - Types of coordinate systems:
    - —Projected
    - —Geodetic
- Geodetic coordinate system concepts:
  - Geodetic optimized rectangles
  - `CS_SRS` dictionary view
  - Restrictions in geodetic coordinate systems
- Whole earth geometry model and tolerance
- Coordinate system transformations
- Units supported by Oracle Spatial

ORACLE

# Geodetic Coordinate System: Concepts

- Any two points on the Earth's surface can be connected using an infinite number of lines.
- The line that defines the shortest distance connecting two points is called a geodesic.
- Consider the two points (180,60) and (0,60):
  - The shortest path connecting these two points does not follow the 60-degree latitude line.
  - It goes through the North Pole.




ORACLE



## Geodetic Coordinate System: Concepts

In geodetic space, there is an infinite number of lines that can be drawn connecting any two points on the Earth's surface.

Given any two points on the Earth's surface, the shortest distance between those points is called a *geodesic*.

If the Earth were a perfectly symmetrical geometry, a geodesic always goes around the Earth and reconnects at the same two points and divides the Earth exactly in half.

The geodesic used to connect points does not necessarily follow longitude or latitude lines.

**Note:** The shortest distance between two points on the Earth's surface can never be along the latitude. The only exception to this is when the two points are on the equator. Two points on the same longitude is always geodesic.

# Geodesic for Antipodal Points

- Typically, only one geodesic connects any two given points on the Earth's surface.
- Antipodal points are points on the exact opposite sides of the Earth's surface.
- Antipodal points can be connected with one of two possible geodesics, each of which follows lines of longitude.
- Adjacent points in a line string or polygon cannot be antipodal; Oracle Spatial does not know which way they should be connected.

ORACLE



## Geodesic for Antipodal Points

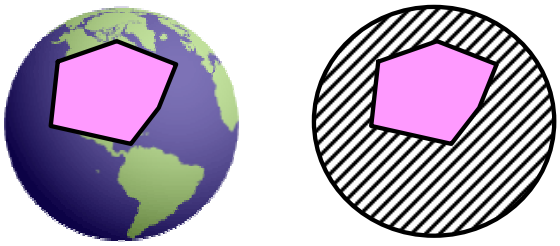
Typically, only one geodesic connects two points on the Earth's surface.

Antipodal points, or points exactly opposite each other on the Earth's spherical surface, can be connected using two geodesics (except when they occur at the North Pole and South Pole, where they can be connected using an infinite number of geodesics).

- Two consecutive points of a line string or a polygon must not be antipodal. Oracle Spatial cannot identify which geodesic to use when points are antipodal. Therefore, one or more additional coordinates are required to enable Oracle Spatial to determine which geodesic to use to connect the points.
- If a segment is required in which a geodesic cannot be used to connect adjacent points, simply add one or more coordinates to enable Oracle Spatial to identify the desired path. Each of the coordinates in the line are connected using geodesics, which may not match the geodesic used if no intermediate coordinates were specified.

# Polygons in a Geodetic Coordinate System

- A polygon can be represented by two closed areas on the Earth's surface.
- Oracle Spatial considers the smaller polygon as the intended polygon.
  - A polygon element's area must be less than half the surface area of the Earth.
  - A geometry can contain multiple polygon elements whose total surface area exceeds half the surface area of the Earth.



ORACLE



## Polygons in a Geodetic Coordinate System

In a geodetic coordinate system, a polygon can be represented by two closed areas on the Earth's surface.

In the example illustrated in the slide, the polygon can represent two areas. The smaller area is specified by the light-colored polygon. The larger area is specified by the striped area, which wraps around the Earth. Oracle Spatial always chooses the smaller of the two polygons.

Any single polygon element in Oracle Spatial must be less than half the surface area of the Earth. This is because Oracle Spatial would not be able to choose which side of the polygon area to use.

A geometry in Oracle Spatial can be made up of multiple polygon elements, where each element has an area less than half the Earth's surface, but the geometry area exceeds half the Earth's surface.

# Geodetic Geometry: Example

- Is this a valid polygon?
- Note that every point is on the 60-degree latitude line.

```
SDO_GEOMETRY(2003, 8307,NULL,
  SDO_ELEM_INFO_ARRAY(1, 1003, 1),
  SDO_ORDINATE_ARRAY( -180,60, -160,60, -140,60,
                      -120,60, -100,60,  -80,60,
                      -60,60,  -40,60,  -20,60,
                       0,60,   20,60,    40,60,
                       60,60,   80,60,   100,60,
                       120,60,  140,60,   160,60,
                      -180,60))
```

ORACLE



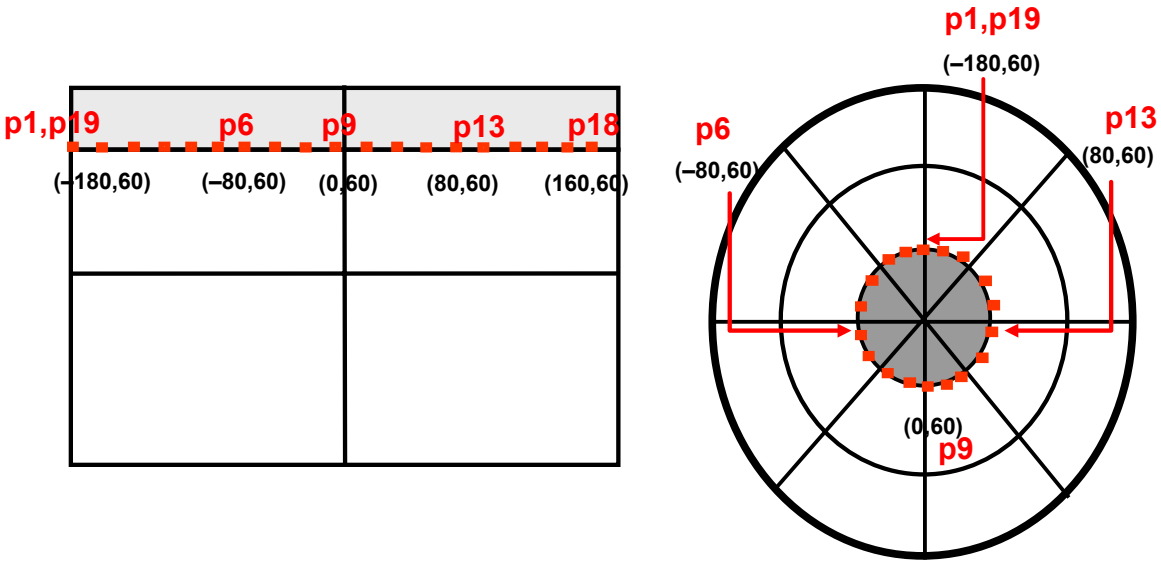
## Geodetic Geometry: Example

In a projected coordinate system, the polygon specified by the code in the slide is not valid. Note that every point is on the 60-degree latitude line. It looks like a straight line.

In geodetic space, the polygon specified is a surface that covers the area north of the 60-degree latitude line.

**Note:** For geodetic geometries, consecutive vertices are connected with a line that follows a geodesic. This is why the geometry in the example in the slide has vertices at every 20 degrees of longitude along the 60-degree latitude line. If desired, adding more vertices along the 60-degree latitude line makes the geometry more accurate.

# Geodetic Geometry: Example




ORACLE



## Geodetic Geometry: Example (continued)

This slide shows how the previous polygon geometry can be displayed in two ways. The first image in the slide is a display of the polygon geometry in an equirectangular viewport. The second image is a display in geodetic space, as viewed from over the North Pole.

Oracle Locator and Oracle Spatial require the elements of a polygon geometry to be oriented. Outer rings must follow a counterclockwise orientation, and inner rings must follow a clockwise orientation.

Imagine walking on the perimeter of an outer ring polygon element. You will be walking in a counterclockwise orientation. The interior portion of the polygon is on your left side.

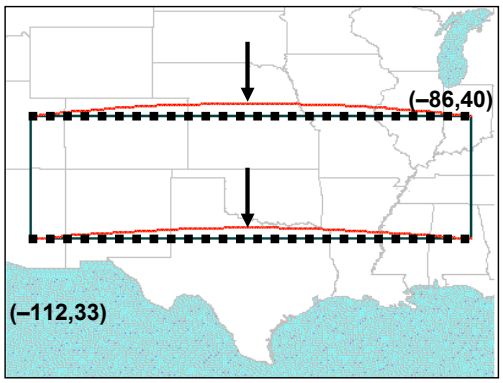
Now, imagine walking on the perimeter of an inner ring polygon element. You will be walking in a clockwise orientation. The interior portion of the polygon is on your left side.

In summary, if the perimeter of an outer ring polygon element is traversed in a counterclockwise direction, or the perimeter of an inner ring polygon element is traversed in a clockwise direction, then the area (or interior region) of the polygon will always be on the left.

**Note:** Polygons can span poles and meridians on the Earth's surface. Such polygons can be stored persistently or can be used as search windows.

# Geodetic Optimized Rectangle Densified Along Latitude Lines

- The user specifies the lower-left and upper-right rectangle vertices.
- Oracle Spatial derives the lower-right and upper-left vertices.
- Lines that follow a longitude line implicitly follow a geodesic.
- Oracle Spatial densifies along latitude lines at one-degree intervals (shown as a dotted black line) instead of following the geodesic (shown as two curved lines, as indicated by the arrows).




ORACLE



## Geodetic Optimized Rectangle Densified Along Latitude Lines

For geodetic optimized rectangles, users specify the lower-left and upper-right vertices. The upper-left and lower-right vertices are derived by Oracle Spatial.

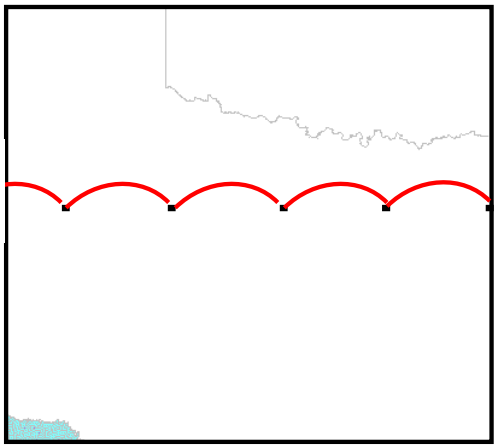
In general, when creating a geometry associated with a geodetic coordinate system, Oracle Spatial considers geodesic distances between consecutive vertices. But the geodetic optimized rectangle is an exception.

The north-to-south line segments in the optimized rectangle run along constant longitude lines. Lines of longitude implicitly follow a geodesic. The desired effect for east-to-west lines is that they follow a constant latitude. Except for along the equator, lines of latitude do not follow a geodesic.

Oracle Spatial densifies the east-to-west lines at one-degree intervals along the latitude so that any east-to-west line stays close to a constant line of latitude.

## Geodetic Optimized Rectangle Densified Along Latitude Lines

Densified line from the previous slide, zoomed in:



- Oracle Spatial still uses geodesics between vertices along the densified line at one-degree intervals.
- If your application requires more accuracy than one-degree intervals, create your own densified polygon (do not use the geodetic optimized rectangle).

ORACLE



### Geodetic Optimized Rectangle Densified Along Latitude Lines (continued)

The internal representation of the geodetic optimized rectangle densifies at one-degree intervals along constant lines of latitude. When performing spatial analysis, Oracle Spatial still considers the geodesics between the densified vertices.

If your application requires that a constant line of latitude be followed more closely than one-degree densification, do not use the geodetic optimized rectangle. Instead, create your own polygon densified along constant latitudes at an interval smaller than one degree.

# Geodetic Optimized Rectangles

- Optimized rectangles are supported with geodetic data:

```
SDO_GEOMETRY(2003,8307,NULL,
  SDO_ELEM_INFO_ARRAY(1,1003,3),
   SDO_ORDINATE_ARRAY(-112,33,-86,40))
```

- Internally, geodetic optimized rectangles:
  - Can span the 180-degree meridian, but not the poles
  - Densify (add coordinates) along the constant latitude in one-degree increments
  - May divide a large optimized rectangle into a multielement geometry, where each element is guaranteed to be smaller than half the Earth's surface area

ORACLE



## Geodetic Optimized Rectangles

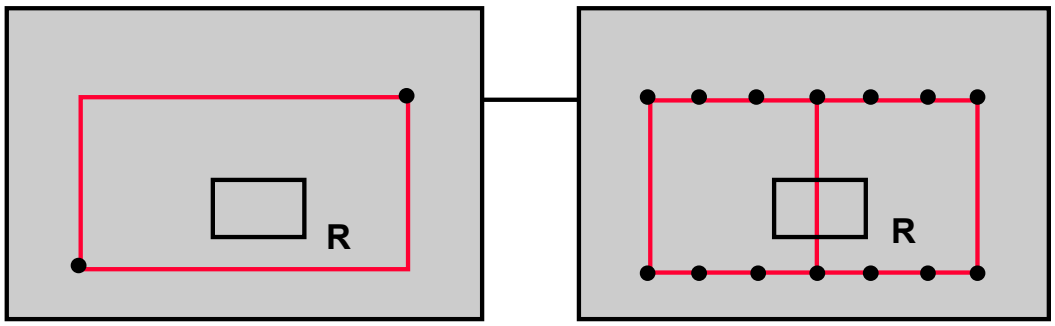
Optimized rectangles are supported with geodetic data. Internally, Oracle Spatial densifies geometries at one-degree intervals along the constant latitude.

Oracle Spatial may divide a single, large optimized rectangle into a multielement geometry, where each element is guaranteed to be smaller than one half of the surface area of the Earth (to conform to the rules of geodetic polygons).

A geodetic optimized rectangle can span the 180-degree meridian, but cannot span the poles. Primarily, geodetic optimized rectangles are used in applications that have the zoom in/zoom out functionality.

# Geodetic Optimized Rectangles

- Geodetic optimized rectangles may be internally represented as a multipolygon.
- Rectangle R is *inside* the first polygon; it is *not inside* the second multipolygon.




Note: This is discussed again when you learn about Spatial queries.

ORACLE



## Geodetic Optimized Rectangles (continued)

This is an example of a possible internal representation of a geodetic optimized rectangle.

Note that the internal representation may be a multipolygon whose edges touch. This is done to ensure that each polygon element of the returned geometry has an area less than half the Earth's surface area.

Because the interior of the original optimized rectangle may get broken up into a geodetic multipolygon geometry, the new geometry must be used only for the following operations:

- As a window for `SDO_FILTER` queries
- As a window for `SDO_RELATE` queries, but only for the `ANYINTERACT` mask
- As a window for the `SDO_ANYINTERACT` operator, which is equivalent to `SDO_RELATE` with the `ANYINTERACT` mask

**Note:** All masks are allowed with a nongeodetic or projected optimized rectangle.

# Coordinate Systems in Oracle Spatial

- The CS_SRS dictionary view contains information about valid coordinate systems.
- Each coordinate system is identified by a unique spatial Reference ID (SRID).

```
DESCRIBE cs_srs
```

```
Name                 Null?          Type
-------------------------------------------------
CS_NAME                             VARCHAR2(68)
SRID                 NOT NULL       NUMBER(38)
AUTH_SRID                           NUMBER(38)
AUTH_NAME                           VARCHAR2(256)
WKTEXT                              VARCHAR2(2046)
CS_BOUNDS                           MDSYS.SDO_GEOMETRY
```

ORACLE



## Coordinate Systems in Oracle Spatial

Oracle Spatial provides over four thousand predefined coordinate systems. The CS_SRS Oracle dictionary view stores definitions of all coordinate systems.

Information about each coordinate system is encoded using the notation defined by the Open Geospatial Consortium (OGC). This information is stored in the WKTEXT column of the CS_SRS dictionary view. Each coordinate system in Oracle Spatial is assigned a unique number, which is stored in the SRID column.

**Note:** The MDSYS Oracle user has granted select access on CS_SRS to PUBLIC.

## CS_SRS View

- The SRID column contains a unique spatial reference ID number.
- The WKTEXT column contains the Well-Known Text description:
  - Geodetic SRIDs when the WKTEXT column begins with "GEOGCS"
  - Projected SRIDs when the WKTEXT column begins with "PROJCS"
- The CS_NAME column contains a short name for the coordinate system.
- The AUTH_NAME column contains the authority name of the coordinate system. This is set to "Oracle" for all rows.
- The AUTH_SRID field currently contains the SRID.
- The CS_BOUNDS field is NULL for all rows.

ORACLE



### CS_SRS View

There are six fields in the CS_SRS view:

- The SRID field contains the unique spatial reference ID number.
- The WKTEXT column holds the Well-Known Text as described by the OGC.
- The CS_NAME field holds a unique short text description of each coordinate system.
- The AUTH_NAME field holds the authority name of the creator of the SRID. In Oracle Spatial, AUTH_NAME is "Oracle." It may be used along with SRID as a global identifier for Oracle Spatial coordinate systems.
- The AUTH_SRID field is a number that can be used to indicate how the entry was derived.
- The CS_BOUNDS field is set aside for Oracle use and is NULL for all rows. It eventually contains a geometry specified in WGS-84 (longitude/latitude), which is the valid polygon boundary for each projection. It remains NULL for geodetic data.

**Note:** GEODETIC_SRIDS is a view that lists SRID values for all geodetic coordinate systems.

# Common Coordinate Systems

Some common coordinate systems include:

- Geodetic
  - NAD 83 (SRID: 8265)
  - WGS 84 (SRID: 8307)
- Projected
  - Continental, country-specific, or state plane projections
    - 32769 Equal-Area Projection (Australia)
    - 32771 Equal-Area Projection (Europe)
    - 32774 Equal-Area Projection (North America)
    - 32770 Equal-Area Projection (China)
    - 32772 Equal-Area Projection (India)
    - 32775 Equal-Area Projection (United States)
    - 81996 Delaware State Plane (meters)

ORACLE



## Common Coordinate Systems

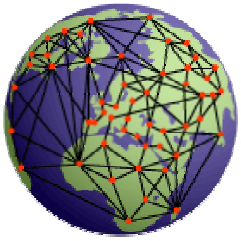
Some common geodetic coordinate systems are:

- North American Datum 1983 (NAD 83)
- World Geodetic System 1984 (WGS 84)

There are a large number of projected coordinate systems used throughout the world. The most accurate projected coordinate systems describe smaller areas. In the United States, there are a number of state plane projected coordinate systems. Smaller states may require only one projected coordinate system, and larger states may require more projected coordinate systems to cover the state with the same amount of precision. Typically, the larger the area covered by a projected coordinate system, the larger the potential for inaccurate spatial analysis.

# Restrictions in a Geodetic Coordinate System

- Element types not supported:
  - Circular arcs or elements that include arcs (compound elements)
  - Circles
- Element type restrictions:
  - No two adjacent points in a line or polygon can be antipodal (exactly on opposite sides of the Earth).
  - No polygon element can have an area that is equal to or greater than half the area of the Earth.



ORACLE



## Restrictions in a Geodetic Coordinate System

There are some restrictions when you use geodetic coordinate systems.

Restrictions in geometric primitive types include:

A geometry cannot contain any arcs. For example, line strings or polygons composed of circular arcs (interpretation 2), compound line strings and polygons that have subelements that are circular arcs, and optimized circles are not supported. If you have geometries containing circles or circular arcs in a projected coordinate system, you can densify them before transforming them to geodetic coordinates, and then perform spatial operations on the resulting geometries.

Additionally, there are restrictions on certain elements:

- Two adjacent points in a line or polygon cannot be antipodal—that is, they cannot be exactly opposite each other on the surface of the Earth.
- The area of a polygon element cannot be equal to or greater than half the area of the Earth. If you need to work with larger elements, first break these elements into multiple smaller elements and work with them. For example, you cannot create a geometry representing the entire land surface of the Earth; however, you can create multiple geometries, each representing a part of the overall land surface. To work with a line string that is greater than or equal to half the perimeter of the Earth, you can add one or more intermediate points on the line so that all adjacent coordinates are less than half the perimeter of the Earth.

# Lesson Agenda

- Coordinate systems:
  - Types of coordinate systems:
    - Projected
    - Geodetic
- Geodetic coordinate system concepts:
  - Geodetic optimized rectangles
  - `CS_SRS` dictionary view
  - Restrictions in geodetic coordinate systems
- Whole earth geometry model and tolerance
- Coordinate system transformations
- Units supported by Oracle Spatial

ORACLE

# Whole Earth Geometry Model

- Provides accurate length, distance, and area calculations on geodetic data
- Supports a large number of length and area units for geodetic data calculations
- Supports geometries that span the 180-degree meridian and poles
- Requires coordinate system bounds to be (–180 to 180) and (–90 to 90)

ORACLE



## Whole Earth Geometry Model

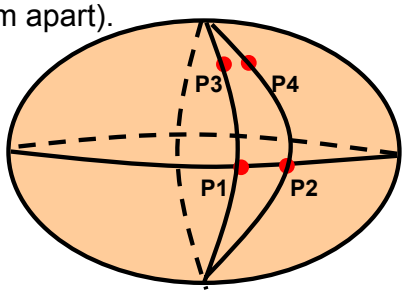
Oracle Spatial returns accurate lengths, areas, and distances for both projected and geodetic data.

The whole Earth geometry model takes into account the curvature of the Earth's surface when performing length, area, and distance calculations on geodetic data. The whole Earth geometry model requires that the coordinate system bounds be set to (–180 to 180) degrees for longitude, and (–90 to 90) degrees for latitude. It can store, index, and operate on geodetic geometries, even if they span the 180-degree meridian and the poles.

Oracle Spatial supports many different length and distance units that are useful for geodetic and projected data, such as foot, meter, and kilometer.

# Tolerance in Projected and Geodetic Coordinate Systems

- Tolerance always has an associated distance unit.
- In projected space, the tolerance has the same unit as the data's coordinate system.
- In geodetic space, the data unit and the tolerance unit are different.
  - Data is in angular units (longitude/latitude).
    - One degree does not consistently refer to any specific distance.
      - P1 and P2 are one degree apart (about 111 km apart).
      - P3 and P4 are one degree apart (about 10 km apart).
  - Tolerance must always be in meters (for example, 0.5 = 1/2 meter resolution).
  - The smallest allowable geodetic tolerance is 0.05 (5 centimeters).




ORACLE



## Tolerance in Projected and Geodetic Coordinate Systems

For projected data, and for data that has no spatial reference ID (`SRID`) value set (no associated coordinate system), the tolerance is still specified in the same coordinate system as the data.

For geodetic data, the data unit and the tolerance unit are not the same. In geodetic space, the data is in angular units of longitude and latitude (sometimes referred to as a decimal degree), but the tolerance value is always in meters.

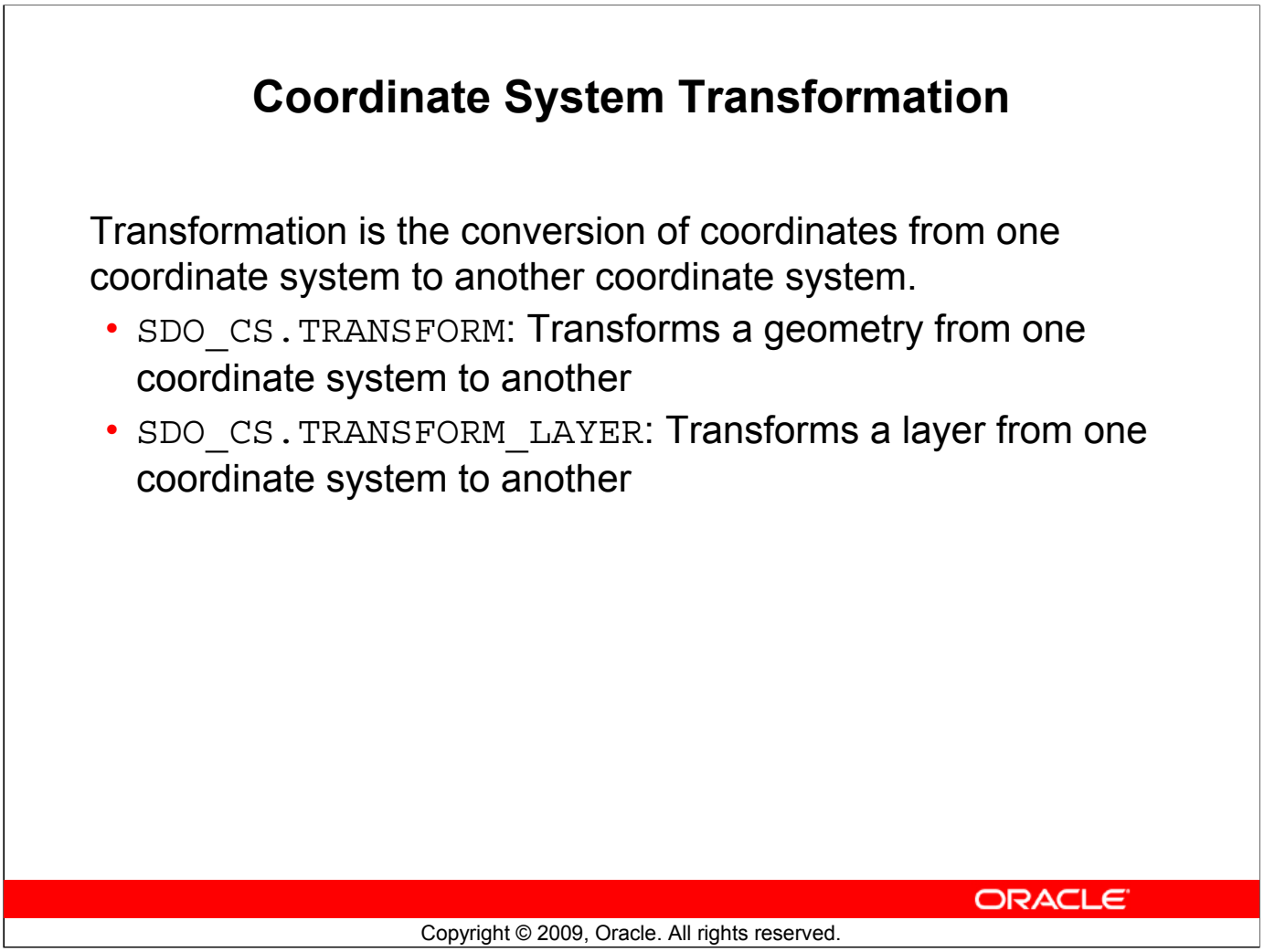
With regards to tolerance, a geodetic coordinate system is different from a projected coordinate system because the distance associated with degrees of longitude varies as the location changes northward or southward. For example, starting at the equator, as you move a degree of latitude towards a pole, the distance it represents gets smaller and smaller until it is 0 kilometers at the pole. A degree of longitude at the equator is approximately 111 kilometers.

Specifying a tolerance value of "1" would have no consistent meaning if the tolerance is specified in angular units. For geodetic data, Oracle Spatial requires that tolerance be specified in meters, so the tolerance value is consistent no matter where the data is located on the Earth.

# Lesson Agenda

- Coordinate systems:
  - Types of coordinate systems:
    - Projected
    - Geodetic
- Geodetic coordinate system concepts:
  - Geodetic optimized rectangles
  - `CS_SRS` dictionary view
  - Restrictions in geodetic coordinate systems
- Whole earth geometry model and tolerance
- Coordinate system transformations
- Units supported by Oracle Spatial

ORACLE




## Coordinate System Transformation

Coordinate system transformation functions are used to transform spatial data from one coordinate system to another. There are many reasons to transform geometries. For example, if a set of data is received in one coordinate system but the rest of the data in use at a site is in a different coordinate system, and common storage is required, coordinate system transformations would be used. Another example is if a data set is in longitude/latitude, but there is a projected coordinate system that better suits the requirements of the application, coordinate system transformations can be used to move the data to the more applicable projection.

Oracle Spatial has the following two coordinate system transformation functions:

- `SDO_CS.TRANSFORM` works with a single geometry at a time, and transforms that geometry from one coordinate system to another.
- `SDO_CS.TRANSFORM_LAYER` takes all the geometries in a layer and transforms the data from one coordinate system to another.

## SDO_CS.TRANSFORM Function

```
SDO_GEOMETRY := SDO_CS.TRANSFORM (<geom>, <to_srid>)
```

- <geom>:
  - Is a geometry of the SDO_GEOMETRY type
  - Can be a variable or a table column
- <to_srid>:
  - Is the spatial reference system ID to transform to
- Returns the SDO_GEOMETRY object

ORACLE





### SDO_CS.TRANSFORM Function

This function accepts as input a single geometry of the SDO_GEOMETRY type and transforms it to the coordinate system specified by <to_srid>.

The input geometry can be a variable, a constructor, a geometry from a table, or the return value from another function. The input geometry must have the SDO_SRID field set with an SRID value from the CS_SRS table. The transform function transforms data only between georeferenced coordinate systems, or between local (non-Earth) coordinate systems. It cannot transform data from a local (non-Earth) coordinate system to a georeferenced coordinate system, or vice versa.

The SDO_CS.TRANSFORM function returns a geometry object that has been transformed to the new coordinate system.

## SDO_CS.TRANSFORM: Example

Transforms the coordinate system of the geometry representing the Hillsborough County in New Hampshire to the coordinate system with SRID 82151

```
SELECT sdo_cs.transform (geom, 82151)
FROM geod_counties
WHERE county = 'Hillsborough'
  AND state = 'New Hampshire';
```

**Note:**

- **All transformations require a valid SDO_SRID field set in the source geometry.**
- **82151 = "New Hampshire 2800 (1983, meters)"—State Plane CS 1983**

ORACLE



### SDO_CS.TRANSFORM: Example

In this example, Hillsborough County in New Hampshire is transformed from a geodetic coordinate system to "New Hampshire 2800 (1983, meters)"—US State Plane Coordinate System 1983.

**Note:** The source geometry used to store Hillsborough County must have the SDO_SRID field set.

## SDO_CS.TRANSFORM_LAYER Procedure

```
SDO_CS.TRANSFORM_LAYER(<table_name>,
                       <geom_column_name>,
                       <destination_table>,
                       <to_srid>)
```

- <table_name>: Name of the table containing the spatial layer
- <geom_column_name>: Name of the column of the SDO_GEOMETRY type
- <destination_table>: Name of the new table created by the procedure
- <to_srid>: Spatial reference ID to transform the coordinate system to

ORACLE



### SDO_CS.TRANSFORM_LAYER Procedure

The SDO_CS.TRANSFORM_LAYER function takes as input the name of a spatial layer, and transforms all geometries to a different coordinate system. It puts the resulting geometries in a new table along with a ROWID pointer back to the original geometry.

<table_name> and <geom_column_name> uniquely identify the layer to be transformed.

<destination_table> is the name of the table created by the procedure, into which the transformed geometries are written.

<to_srid> is the spatial reference ID to transform the layer to.

**Note:** This procedure requires that the input layer have a valid SDO_SRID value in each geometry and in the SRID field of USER_SDO_GEOM_METADATA.

Transformations can occur only between two georeferenced coordinate systems or two local (non-Earth) coordinate systems.

## SDO_CS.TRANSFORM_LAYER: Example

- This transforms the geometries in the GEOM column to the projected "Equal-Area Projection (United States)" coordinate system (SRID = 32775) geometries.
- Transformed geometries are stored in the new PROJ_COUNTIES table that is created by the procedure.
- The PROJ_COUNTIES table consists of two columns:
  - The GEOMETRY column of the SDO_GEOMETRY type
  - The SDO_ROWID column that stores the pointer to the original geometry

```
begin
   SDO_CS.TRANSFORM_LAYER (
     'GEOD_COUNTIES', 'GEOM', 'PROJ_COUNTIES', 32775);
end;
/
```

ORACLE



### SDO_CS.TRANSFORM_LAYER: Example

In this example, the SDO_CS.TRANSFORM_LAYER procedure is executed. It takes the GEOM column (of the SDO_GEOMETRY type ) from the GEOD_COUNTIES table and transforms the geometries from the geodetic "Longitude / Latitude (WGS 84)" coordinate system (SRID = 8307) to the projected "Equal-Area Projection (United States)" coordinate system (SRID = 32775).

The transformed geometries are written into a table created by the SDO_CS.TRANSFORM_LAYER procedure. The table name is specified as PROJ_COUNTIES. The PROJ_COUNTIES table has two columns: a column called GEOMETRY of the SDO_GEOMETRY type, which contains transformed geometries, and a column called SDO_ROWID, which contains a pointer to the original geometry. When the transformation is complete, attribute data can be copied from the source table to the target table (based on SDO_ROWID), or the geometry column can be copied from the target table to the source table (also based on SDO_ROWID).

**Note:** The D:\labs\demo\transform_layer_example.sql script transforms a layer from one coordinate system (8307) to a different coordinate system [in this case, 32775, an Equal-Area Projection (United States)].

# Lesson Agenda

- Coordinate systems:
  - Types of coordinate systems:
    - —Projected
    - —Geodetic
- Geodetic coordinate system concepts:
  - Geodetic optimized rectangles
  - `CS_SRS` dictionary view
  - Restrictions in geodetic coordinate systems
- Whole earth geometry model and tolerance
- Coordinate system transformations
- Units supported by Oracle Spatial

ORACLE

# Units Supported by Oracle Spatial

- Every coordinate system has an associated unit defined in its `WKTEXT` description.
- For spatial operators and functions with a `UNIT` parameter:
  - The default unit for projected data is the unit associated with the projected coordinate system
  - The default unit for geodetic data is meters
  - The default unit for the local coordinate system is the unit specified in the local coordinate system

ORACLE





## Units Supported by Oracle Spatial

The `UNIT` parameter is supported in Oracle Spatial for functions and operators that require or return length, area, or distance information. Operators that have a `UNIT` parameter include `SDO_WITHIN_DISTANCE` and `SDO_NN_DISTANCE`. Functions that have a `UNIT` parameter include `SDO_LENGTH`, `SDO_DISTANCE`, `SDO_AREA`, and `SDO_BUFFER`. You learn about these operators and functions in the following lessons.

The `UNIT` parameter is available only when a valid `SDO_SRID` is specified in the geometry.

Every coordinate system has an associated unit defined in its WKT description in the `WKTEXT` column of the `CS_SRS` table.

All operations in Oracle Spatial have a default unit associated with them.

- For projected data, the default unit is the unit associated with the projected coordinate system itself. For most projected coordinate systems, this is either foot or meter.
- For geodetic data, the default unit is meter.
- For local (non-Earth) coordinate systems, the default is the unit specified for the local (non-Earth) coordinate system.

## Units Supported by Oracle Spatial

- The SDO_DIST_UNITS dictionary view contains one row for each valid *distance* unit specification.
- The SDO_AREA_UNITS dictionary view contains one row for each valid *area* unit specification.
- The SDO_ANGLE_UNITS dictionary view contains one row for each valid *angle* unit specification such as degree, radian, and so on.
- Users have read access to views.

```
SELECT sdo_unit FROM sdo_dist_units;
```

```
 SDO_UNIT
--------------------------------
KILOMETER
CENTIMETER
NAUT_MILE
…
```

ORACLE



### Units Supported by Oracle Spatial (continued)

Distance units include:

METER, KILOMETER, CENTIMETER, MILLIMETER, MILE, NAUT_MILE, SURVEY_FOOT, FOOT, INCH, YARD, CHAIN, ROD, LINK, MOD_USFT, CL_F, IND_FT, LINK_BEN, LINK_SRS, CHN_BEN, CHN_SRS, IND_YARD, SRS_YARD, and FATHOM

Area units include:

SQ_METER, SQ_KILOMETER, SQ_CENTIMETER, SQ_MILLIMETER, SQ_CHAIN, SQ_FOOT, SQ_INCH, SQ_LINK, SQ_MILE, SQ_ROD, SQ_SURVEY_FOOT, SQ_YARD, ACRE, HECTARE, PERCH, and ROOD

Angle units include:

DEGREE, RADIAN, SECOND, MINUTE, GON, GRAD, DEGREE MINUTE, DEGREE HEMISPHERE, HEMISPHERE DEGREE, DEGREE MINUTE HEMISPHERE, HEMISPHERE DEGREE MINUTE, HEMISPHERE DEGREE MINUTE, SEXAGESIMAL DMS.S, and DEGREE (SUPPLIER TO DEFINE REPRESENTATION)

The SDO_UNIT column in the SDO_DIST_UNITS view has a list of the length and distance units that are supported. The SDO_UNIT column in the SDO_AREA_UNITS view has a list of the area units that are supported. Similarly, the SDO_UNIT column in the SDO_ANGLE_UNITS view has a list of the angle units that are supported.

# Summary

In this lesson, you should have learned how to:

- Define coordinate systems
- Explain the difference between georeferenced and nongeoreferenced coordinate systems
- Describe the concepts of geodetic coordinate systems
- Define the whole Earth geometry model
- Describe the coordinate system transformation functions
- Identify units supported by Oracle Spatial

ORACLE



## Summary

In this lesson, you got an overview of the concepts related to coordinate systems. This lesson described the concepts related to geodetic coordinate systems in detail. This lesson also discussed the transformation functions and the units that are supported by Oracle Spatial.

# Practice 4: Overview

This practice covers running the transformation functions.

ORACLE



## Practice 4: Overview

In this practice, you run the coordinate system transformation routines on geometries. The routines transform the geometries into a new coordinate system.

## Practice 4

1. Describe the `CS_SRS` dictionary view.
2. Run the `lab_04_02.sql` script located in the `D:\labs\labs` folder. This script creates the `geom_shapes` table and inserts two geometries with the `SRID` 8307.
3. View the two inserted geometries. The spatial column in the table, `geom_shapes`, is `shape`.
4. Insert a new row for `geom_shapes (shape)` in the `USER_SDO_GEOM_METADATA` view.
   i. Set tolerance to `0.05`.
   ii. Set the `Long` axis range: `-180` to `180`
   iii. Set the `Lat` axis range: `-90` to `90`
   iv. Set `SRID` to `8307`.
5. Transform the geometry with Object_id as `1` in the `geom_shapes` table to `32618 (WGS 84/UTM zone 18N)`.
   **Note:** In this transformation, the original geometry is not actually transformed. The transformed geometry is displayed as a result.
6. Transform the entire shape layer and put the results in the table named `geom_shapes_cs_32618`.
   **Note:** This transformation procedure actually transforms the geometries into the new coordinate system and stores the results in the new table, `geom_shapes_CS_32618`. The new table is created by the procedure itself.
7. Describe the `geom_shapes_CS_32618` table.
8. Select all the geometries from the `geom_shapes_CS_32618` table.

# 5

# Loading Spatial Data

# Objectives

After completing this lesson, you should be able to do the following:

- Identify the methods for loading spatial data
- Create the control files associated with the bulk-loading of spatial data by using SQL*Loader
- Export and import spatial data
- Use the Java Shapefile Converter tool to load an Environmental Systems Research Institute (ESRI) shapefile into the Oracle Spatial format

ORACLE



## Objectives

This lesson explains how spatial data is loaded into Oracle Spatial. It describes how to load data by using the standard Oracle utilities and commands (SQL*Loader, Data Pump `impdp`, `INSERT`, and so on), as well as how to use a tool to move data from the Environmental Systems Research Institute (ESRI) shapefiles into the Oracle Spatial format.

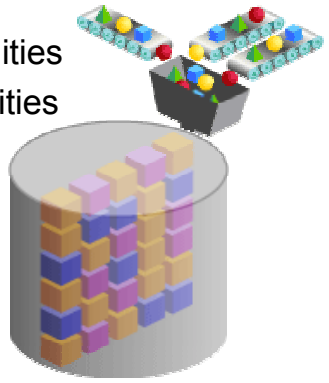
# Lesson Agenda

- Different ways of loading spatial data
- Loading spatial data by using SQL*Loader
- Export and import utilities of the Data Pump technology
- Transportable tablespace
- Transactional insert
- Java Shapefile Converter

# Loading Spatial Data

Two ways of loading spatial data:

- Bulk-loading:
  - SQL*Loader
  - Export and import
    - Using Data Pump export and import utilities
    - Using the original export and import utilities
  - Transportable tablespace
- Transactional inserts:
  - `INSERT` statement



ORACLE



## Loading Spatial Data

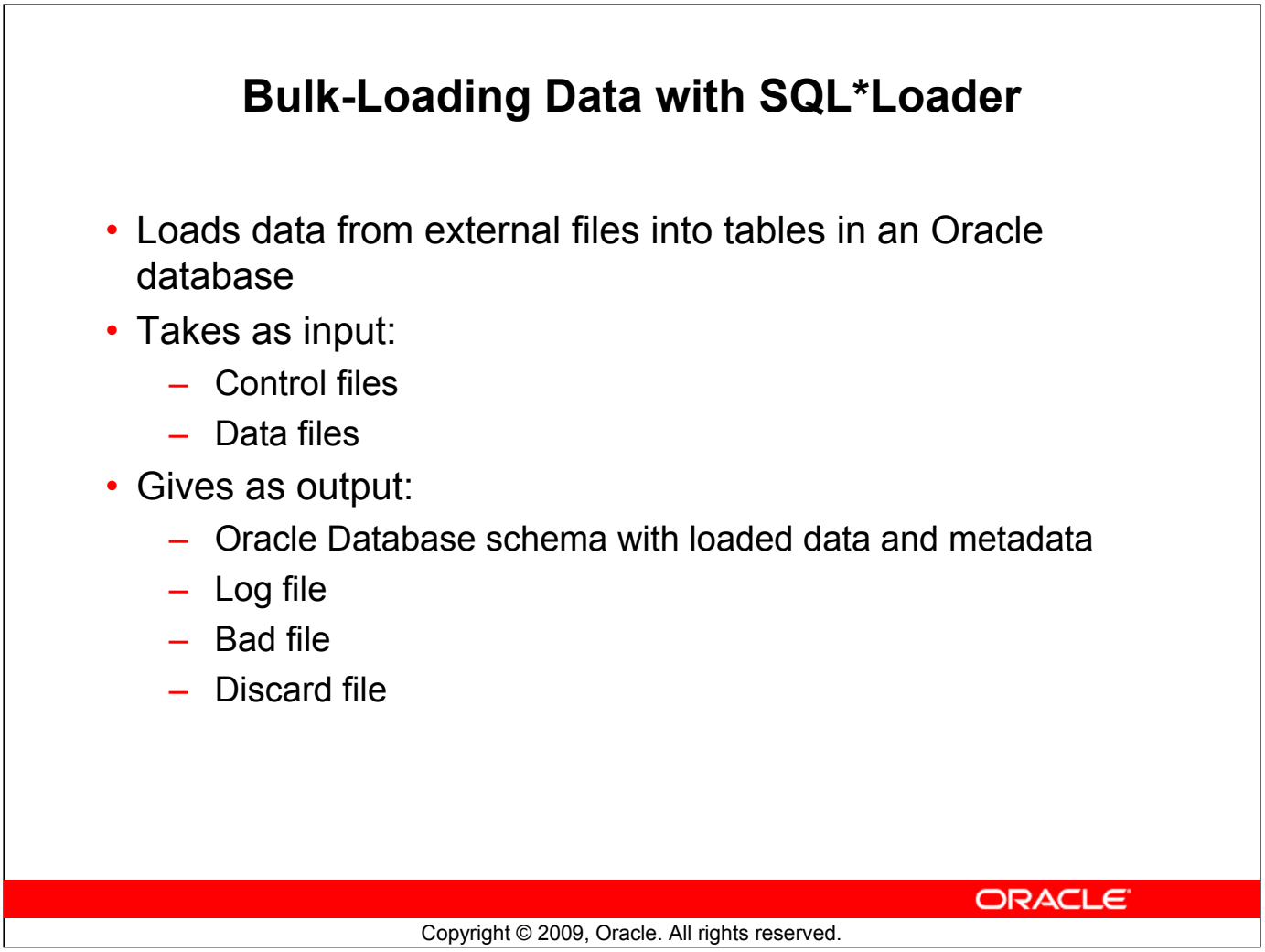
Two steps are involved in moving data into a database so that it can be queried efficiently: loading data tables, and creating or updating the index on the tables. Data stored in Oracle Spatial is treated in the same manner as nonspatial data.

Spatial data can be loaded in two ways:

- **Bulk-loading:** This can be accomplished with different utilities, such as SQL*Loader, Data Pump export and import utilities, traditional dump file export and import utilities, or third-party tools (such as the Feature Manipulation Engine [FME]). The import utility moves data from one Oracle database to another Oracle database. No new syntax is required for moving spatial data with the Oracle database's import and export utilities. SQL*Loader loads plain ASCII files into the Oracle database. Because SQL*Loader can load Oracle objects into the database, it can load `SDO_GEOMETRY` too.
- **Transactional inserts:** This process is used to insert relatively small amounts of data into the database, and is analogous to the `INSERT` command in SQL.

# Lesson Agenda

- Different ways of loading spatial data
- Loading spatial data by using SQL*Loader
- Export and import utilities of the Data Pump technology
- Transportable tablespace
- Transactional insert
- Java Shapefile Converter




## Bulk-Loading Data with SQL*Loader

You can use bulk-loading to add large amounts of data (spatial and nonspatial) into the Oracle database. Bulk-loading is accomplished by inserting data into the table by using the Oracle SQL*Loader utility.

SQL*Loader loads data from external files into tables in an Oracle database. SQL*Loader accepts input data in a variety of formats, can perform filtering (selectively loading records based on their data values), and can load data into multiple Oracle database tables during the same load session.

SQL*Loader takes a control file as its input, which describes the load to SQL*Loader. The control file also specifies the input data files. A parameter file can be specified, which provides a mechanism to supply additional command-line parameters to SQL*Loader.

As it executes, SQL*Loader produces a log file, where it writes information about the load. If records are rejected (typically because of incorrect data), it produces a bad file containing the rejected records. It may also produce a discard file containing records that did not meet the specified selection criteria. The SQL*Loader command-line executable name may differ based on the operating system platform.

For more information about SQL*Loader, refer to the *Oracle Database Utilities* documentation.

# SQL*Loader Control and Data Files

Control file (.ctl)

```
LOAD DATA
 INTO TABLE cities
 FIELDS TERMINATED BY '|'
 TRAILING NULLCOLS (
   CITY NULLIF CITY = BLANKS,
   STATE_ABRV NULLIF STATE_ABRV = BLANKS,
   POP90,
   RANK90,
   LOCATION COLUMN OBJECT
   (
     SDO_GTYPE        INTEGER EXTERNAL,
     SDO_SRID         CONSTANT 8307,
     SDO_POINT COLUMN OBJECT
       (X             FLOAT EXTERNAL,
        Y             FLOAT EXTERNAL)
   ))
```

Data file (.dat)

```
New York|NY|7322564|1| 2001|-73.943849000|40.669800000|
Los Angeles|CA|3485398|2| 2001|-118.411201000|34.112101000|
Chicago|IL|2783726|3| 2001|-87.684965000|41.837050000|
Houston|TX|1630553|4| 2001|-95.386728000|29.768700000|
Philadelphia|PA|1585577|5| 2001|-75.134678000|40.006817000|
San Diego|CA|1110549|6| 2001|-117.135770000|32.814950000|
```

ORACLE



## SQL*Loader Control and Data Files

In the example in the slide, a SQL*Loader control file is used for loading point data into Oracle Spatial. Note the following:

- `INTO TABLE cities` means that the `cities` table is to be loaded.
- `FIELDS TERMINATED BY '|'` means that each field will be ended with a vertical bar (the fields are not of fixed length).
- `TRAILING NULLCOLS` sets to `NULL` any columns specified if the input record ends before all the fields have been filled.
- Each column is specified next. `NULLIF` column=`BLANKS` sets the column to `NULL` if only blanks (spaces) are being loaded in the column.
- The location is a column of the `SDO_GEOMETRY` type. The syntax shown (that is, the `COLUMN OBJECT` keywords) is specific for objects; the syntax is not specific to Oracle Spatial.
- The `EXTERNAL` keyword with the data type is required.

Parts of the `SDO_GEOMETRY` object such as `SDO_ELEM_INFO` and `SDO_ORDINATES` fields that are not specified in the definition, are set to `NULL`.

# Loading Lines and Polygons by Using SQL*Loader

Control file (.ctl)

```
LOAD DATA
 CONTINUEIF NEXT(1:1) = '#'
 INTO TABLE counties
 FIELDS TERMINATED BY '|'
 TRAILING NULLCOLS (
  COUNTY NULLIF COUNTY = BLANKS,
  STATE NULLIF STATE = BLANKS,
  POPPSQMI,
  GEOM COLUMN OBJECT
   (SDO_GTYPE     INTEGER EXTERNAL,
    SDO_SRID      CONSTANT 8307,
    SDO_ELEM_INFO VARRAY TERMINATED BY '|/' (X FLOAT EXTERNAL),
    SDO_ORDINATES VARRAY TERMINATED BY '|/' (X FLOAT EXTERNAL) ))
```

Data file (.dat)

```
 Autauga|Alabama|57.428300000|
#2003|1|1003|1||/
#-86.916969000|32.664028000|-86.816589000|32.659988000|-86.713409000|....|
#-87.765160000|31.297176000|-86.916969000|32.664028000||/
 Baldwin|Alabama|61.569000000|
#2003|1|1003|1||/
#-87.765160000|31.297176000|-87.760429000|31.297289000|-87.759232000|....||/
```

ORACLE





## Loading Lines and Polygons by Using SQL*Loader

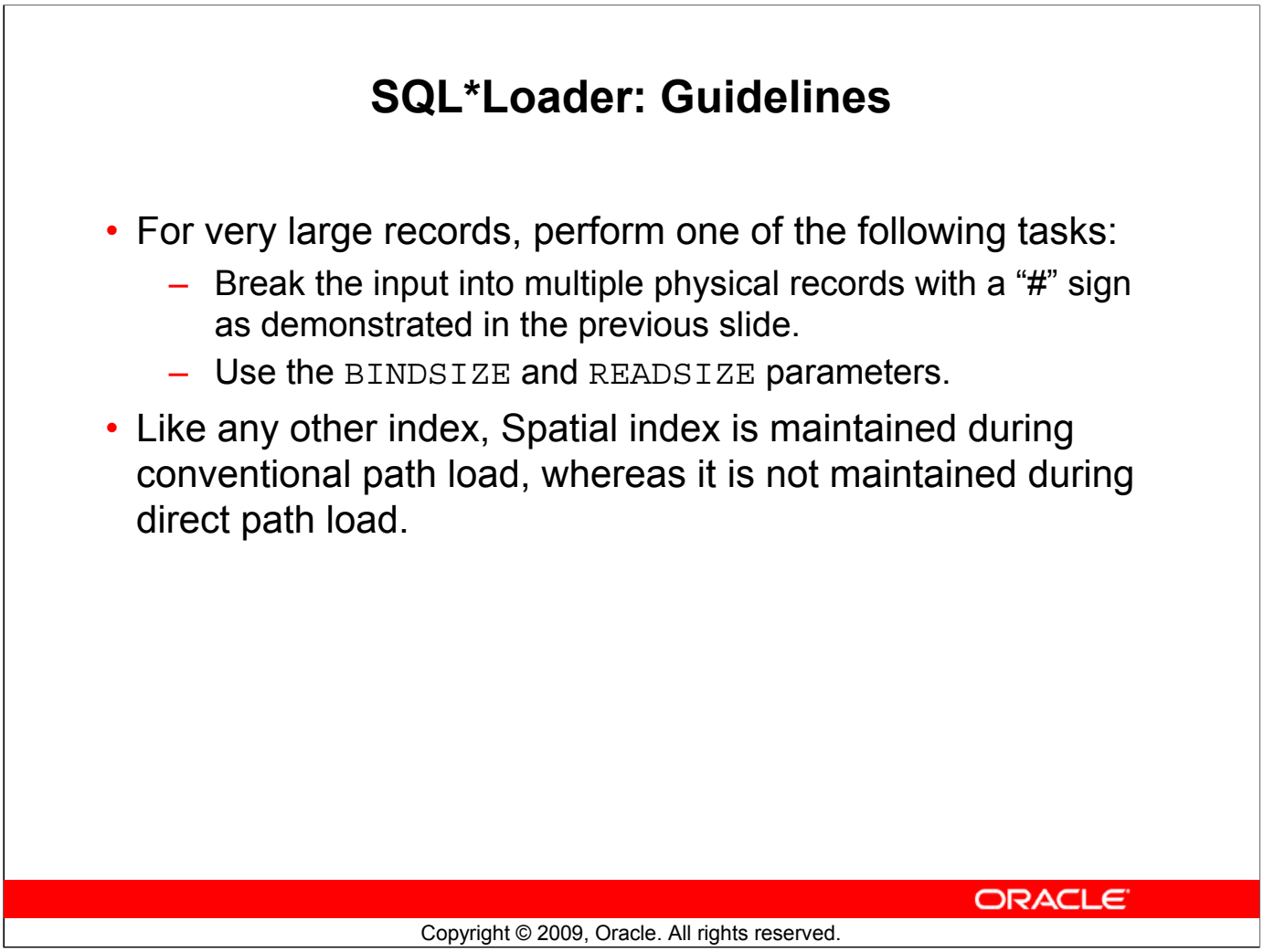
Note the `CONTINUEIF NEXT(1:1) = '#'` clause in the slide example. This syntax causes SQL*Loader to look at the first character of each physical input record, and if it is #, it continues the logical record from the previous line. One of the requirements when you use this syntax is that there should *never* be data in the first position of a physical record. SQL*Loader always looks at the first character position, and if it is a #, it treats the record as a continuation of the previous record. If there is no #, it treats everything *after* the first character as the start of a new record to be loaded.

This example loads the `counties` table, and the fields in this table are terminated by the vertical bar (|).

In this example, the `VARRAY` fields (`SDO_ELEM_INFO` and `SDO_ORDINATES`) within `SDO_GEOMETRY` are being loaded (the previous example loaded the `SDO_POINT` field and left `SDO_ELEM_INFO` and `SDO_ORDINATES` as `NULL`). There is a special syntax to end each of the varray types (each value within the varray is terminated with a vertical bar, but the varray itself also needs to be terminated). The following syntax is the way to describe to SQL*Loader that the `SDO_ELEM_INFO` varray ends with `'|/'`:

```
SDO_ELEM_INFO VARRAY TERMINATED BY '|/' (X FLOAT EXTERNAL)
```

The same syntax is shown for the `SDO_ORDINATES` varray.




## SQL*Loader: Guidelines

You should consider the following guidelines when loading a spatial layer by using SQL*Loader:

- For very large records:
  - Break the record into two or more physical records. The example in the previous slide shows how to break up physical records by using the `CONTINUEIF` syntax.
  - Use the `BINDSIZE` and `READSIZE` parameters to enable SQL*Loader to use large records

Spatial indexes are maintained when using the conventional path mode in SQL*Loader, whereas it is not maintained during direct path loading. It is recommended that you drop the spatial indexes before the load because this can slow down the bulk-loading of large amounts of spatial data significantly. Re-create spatial indexes after the load.

# Limitations of SQL*Loader

- It cannot load directly from common spatial formats or vendor-specific data files.
- Use third-party tools, typically provided by the GIS or spatial software vendor.
  - Feature Manipulation Engine (FME) from Safe Software (www.safe.com) can load many formats into Oracle Spatial.
- Use Oracle's Java-based Shapefile Converter for the Environmental Systems Research Institute (ESRI) shapefile format.

ORACLE



## Limitations of SQL*Loader

SQL*Loader does not recognize proprietary spatial data formats of other vendors, so you cannot use SQL*Loader to add other vendors' spatial data directly into an Oracle database.

To load non-Oracle spatial data into Oracle, there are three choices:

- Use third-party tools to move non-Oracle spatial data into Oracle. One such tool is Feature Manipulation Engine (FME) from Safe Software (www.safe.com).
- If the Geographic Information System (GIS) or spatial software vendor supports its own format as well as the Oracle Spatial format, you can use the tools provided by that vendor to move data from its format to the Oracle Spatial format.
- Use Oracle's Java-based Shapefile-to-Spatial converter. If your data can be extracted in the ESRI shapefile format, the converter can be invoked to load that spatial data, along with its associated attributes, into Oracle.

# Lesson Agenda

- Different ways of loading spatial data
- Loading spatial data by using SQL*Loader
- Export and import utilities of the Data Pump technology
- Transportable tablespace
- Transactional insert
- Java Shapefile Converter

# Export and Import Utilities

- The export utility is used for bulk-unloading of data.
- The import utility is used for bulk-loading of data.
- The export utility extracts data into a dump file.
- The import utility reads the data that was exported back into the database.
- Use either the export and import utilities invoked by the `exp` and `imp` commands respectively or the Data Pump export and import utilities invoked by the `expdp` and `impdp` commands respectively (recommended).

ORACLE





## Export and Import Utilities

The export and import utilities are used for bulk-unloading (export) and bulk-loading (import) of Oracle tables respectively. There is no Oracle Spatial–specific syntax required for exporting tables with spatial columns.

The export utility extracts data into a dump file. By default, the export utility unloads all data associated with a user's schema. By specifying the `TABLES` parameter, the contents of one or more specific tables can be unloaded. For more information about export and import utilities, refer to the *Oracle Database Utilities* guide.

**Note:** Dump files generated by the Data Pump export utility are not compatible with dump files generated by the original export utility. Therefore, files generated by the original export (`exp`) utility cannot be imported with the Data Pump import (`impdp`) utility. In most cases, Oracle recommends that you use the Data Pump export and import utilities. They provide enhanced data movement performance in comparison to the original export and import utilities.

Oracle Database 10*g* supports the Data Pump export and import utility.

# Using Data Pump Export

- Unloads data and metadata into a set of operating-system files called a dump file set
- Enables high-speed movement of data and metadata from one database to another
- Is invoked using the `expdp` command
- Accepts parameters as command-line arguments or in a parameter file
- Provides different modes to export different portions of the database—for example, full, schema, or table mode

ORACLE



## Using Data Pump Export

Data Pump Export is a utility for unloading data and metadata into a set of operating system files called a dump file set. The dump file set can be imported only by the Data Pump Import utility. The dump file set is written in a proprietary, binary format and is made up of one or more disk files that contain table data, database object metadata, and control information. During import, Data Pump Import uses these files to locate each database object in the dump file set.

Because the dump files are written by the server, rather than by the client, the database administrator (DBA) must create directory objects. Data Pump Export is invoked using the `expdp` command. The characteristics of the export operation are determined by the export parameters that you specify either on the command line or in a parameter file. Export provides different modes of unloading different portions of the database. The mode is specified on the command line, using the appropriate parameter. A full export is specified using the `FULL` parameter wherein the entire database is unloaded. A table mode export is specified using the `TABLES` parameter wherein only a specified set of tables, partitions, and their dependent objects are unloaded.

# Using Data Pump Export

- Data pump requires you to create a directory for the data files and log files that it will create and read.
- Use the `CREATE DIRECTORY` command to create a directory object pointing to an existing directory in the file system.
- To access the Data Pump file from the directory, you must have read and write privileges on the directory.

```
CREATE DIRECTORY student_dumpdir as 'd:\labs\data';
GRANT READ, WRITE ON DIRECTORY student_dumpdir TO
student;
```

ORACLE



## Using Data Pump Export (continued)

In the slide example, a directory, `student_dumpdir`, is created. The directory on the file system must exist.

The student user is granted read and write privileges on the directory.

**Note:** To be able to create any directory, you must log in as a privileged user with the `ADMIN` role or the DBA must grant you the `CREATE ANY DIRECTORY` privilege.

# Using Data Pump Export

Schema mode (default) export example:

```
expdp student DIRECTORY=student_dumpdir
DUMPFILE=student.dmp
LOGFILE=student_dumpdir:expschema.log
```

ORACLE



## Using Data Pump Export (continued)

In the slide example, the whole student schema is unloaded into a `student.dmp` file that will be created in the location specified by the directory object, `student_dumpdir`. Optionally, you can specify the `LOGFILE` parameter to generate a log file. You can also specify the `CONTENT=DATA_ONLY` parameter to indicate that you want to unload only the data and not the metadata. For more information, refer to the *Oracle Database Utilities* documentation.

# Data Pump Export: Maintaining Metadata

- For indexed spatial data, the associated geometry metadata is also unloaded.
- For nonindexed spatial data, only the data for the column is unloaded.
  - Move spatial metadata by creating a table with the contents of `USER_SDO_GEOM_METADATA` and export it to the dump file.

```
CREATE TABLE my_md AS
 SELECT * FROM user_sdo_geom_metadata;
```

ORACLE



## Data Pump Export: Maintaining Metadata

If a spatial column is not indexed, Data Pump Export unloads only the data for the column (not the metadata). If the column is spatially indexed, Export also unloads the contents of the `USER_SDO_GEOM_METADATA` view associated with that table, as well as the command to rebuild the index on import.

You can cause spatial metadata to be exported by creating a table with the contents of the `USER_SDO_GEOM_METADATA` view, and exporting that table along with the other data to be exported.

# Using Data Pump Import

- Loads an export dump file set into a target database
- Is invoked using the `impdp` command
- Accepts parameters as command-line arguments or in a parameter file
- Loads the entire dump file set in the mode in which the export operation was run, if import mode is not specified

```
impdp student DIRECTORY=student_dumpdir
DUMPFILE=student.dmp
```

```
impdp student TABLES=geod_counties
dumpfile=student_dumpdir:student.dmp
logfile=student_dumpdir:student.log
```

Note: The command must be on one line in the command prompt.

ORACLE



## Using Data Pump Import

Data Pump Import is a utility for loading an export dump file set into a target system. Data Pump Import is invoked using the `impdp` command. You can specify the behavior of the import operation by the import parameters that you specify either on the command line or in a parameter file.

When the source of the import operation is a dump file set, specifying a mode is optional. If no mode is specified, Import attempts to load the entire dump file set in the mode in which the export operation was run.

You can specify a full import by using the `FULL` parameter. In full import mode, the entire content of the source (dump file set) is loaded into the target database. This is the default for file-based imports.

A table-mode import is specified using the `TABLES` parameter. In table mode, only the specified set of tables, partitions, and their dependent objects are loaded. The source can be a full, schema, tablespace, or table-mode export dump file set.

**Note:** Even when you import a Data Pump file, you must create a directory pointing to the location where your `.dmp` file is. You must have read and write access to the directory.

For more information, refer to the *Oracle Database Utilities* documentation.

# Data Pump Import: Maintaining Metadata

- If the exported spatial column is indexed:
  - A row is added to the `USER_SDO_GEOM_METADATA` view
  - The index is rebuilt by using the same command as originally specified
  - And the `TABLESPACE` parameter is specified, that tablespace must exist
  - With `impdp`, you can use `REMAP_SCHEMA` when the source schema and target schema names are different
- If the exported spatial column is not indexed:
  - Insert metadata into the `USER_SDO_GEOM_METADATA` view from the exported table containing the metadata

```
INSERT INTO user_sdo_geom_metadata SELECT * FROM my_md;
```

ORACLE



## Data Pump Import: Maintaining Metadata

If the export file contains tables with spatial columns that were indexed, an entry is added to the `USER_SDO_GEOM_METADATA` view and the index is rebuilt using the same command it was originally built with.

If the `TABLESPACE` parameter was used when the index was built on the instance that the database was exported from, that tablespace must exist in the instance that the table is imported into, or the index build will fail.

If the Data Pump `.dmp` file was created using the `student` schema and you want to import the same `.dmp` file to an `instructor` schema, you must use the `REMAP_SCHEMA` parameter. The syntax is:

```
REMAP_SCHEMA=source_schema:target_schema
```

If the spatial column was not indexed, the metadata information must be added manually after the import. If a table with spatial metadata was created, the metadata can be added back into the `USER_SDO_GEOM_METADATA` view by using the `INSERT INTO … SELECT FROM` syntax shown in the slide.

# Lesson Agenda

- Different ways of loading spatial data
- Loading spatial data by using SQL*Loader
- Export and import utilities of the Data Pump technology
- Transportable tablespace
- Transactional insert
- Java Shapefile Converter

# Transportable Tablespaces

- Is a very fast way of moving large volumes of data between two Oracle databases
- Uses the `expdp` and `impdp` utilities
- Requires that a specific setup be performed before and after transporting spatial data with spatial indexes
- Enables you to copy database files from one endian platform to a different endian platform
  - Spatial data can be moved.
  - Spatial indexes cannot be transported across different endian platforms.

Note: For more information about transportable tablespaces, see the lesson titled “Managing Spatial Indexes.”

ORACLE



## Transportable Tablespaces

Transportable tablespaces, an Oracle database feature, enables you to copy database files from one system to another system. This is done using Oracle’s export and import utilities such as `expdp` and `impdp`. The Oracle database enables you to copy database files from one type of endian platform (for example, an Intel processor–based machine) to another type of endian platform (for instance, Sun SPARC). Spatial data can be transported across platforms, but currently spatial indexes cannot be transported across platforms. To see all available platforms and their endian format, enter:

```
SELECT * FROM V$TRANSPORTABLE_PLATFORM;
```

Examples of results:

```
PLATFORM_ID   PLATFORM_NAME            ENDIAN_FORMAT
1             Solaris[tm] OE (32-bit)  Big
10            Linux IA (32-bit)        Little
16            Apple Mac OS             Big
```

When you move spatial data with spatial indexes by using transportable tablespaces, you must perform specific steps before exporting and after importing the tablespace.

# Lesson Agenda

- Different ways of loading spatial data
- Loading spatial data by using SQL*Loader
- Export and import utilities of the Data Pump technology
- Transportable tablespace
- Transactional insert
- Java Shapefile Converter

ORACLE

# Transactional Inserts

- Use the standard SQL `INSERT` statement to add geometric features and attributes to an existing spatial layer.
- A spatial index is automatically updated whenever new data is inserted in a spatial layer.

```
INSERT INTO LINES VALUES (
attribute 1, …. attribute n,
      SDO_GEOMETRY (
            2002, null, null,
            SDO_ELEM_INFO_ARRAY (1,2,1),
              SDO_ORDINATE_ARRAY (10,10, 20,25,
                                  30,10, 40,10))
);
```

ORACLE



## Transactional Inserts

Transactional inserts are used to add geometries to an existing spatial column—for example, a new linear transportation feature, an island polygon, an address location, or a hydrologic facility.

The standard `INSERT` statement is used to add spatial data. You have already inserted many primitive geometry types by using the `INSERT` statement in the practices of the earlier lessons.

Note the use of the geometry constructor. In this case, the geometry constructor is used to create a geometry object. Inside the constructor, there are two other constructors: one for the `SDO_ELEM_INFO` field and one for the `SDO_ORDINATES` field.

If a spatial layer has a spatial index, the index is maintained automatically when new data is inserted into the table (as it is for updates and deletes). More information about spatial indexes is available in the lesson titled “Indexing Spatial Data.”

## Limitations of Using the SDO_GEOMETRY Constructor in the INSERT Statement

- You can insert no more than 999 values in the SDO_ORDINATES field when using the SDO_GEOMETRY constructor directly.
- Use the host variable for the geometry object instead.

```
DECLARE
  geom SDO_GEOMETRY :=
         SDO_GEOMETRY (2003, null, null,
          SDO_ELEM_INFO_ARRAY(1,1003,3),
           SDO_ORDINATE_ARRAY (-109,37,-102,40));
BEGIN
  INSERT INTO TEST_1 VALUES (GEOM);
  COMMIT;
END;
/
```

ORACLE



### Limitations of Using the SDO_GEOMETRY Constructor in the INSERT Statement

When you use geometry constructors in an INSERT statement, there is a limitation of 999 values in the VARRAY constructor. If you need to insert more than 999 values in the SDO_ELEM_INFO or SDO_ORDINATES array, you can load data from within a program or use PL/SQL to load data.

In this example, an anonymous PL/SQL block is shown that can insert a geometry with a large number of values in the SDO_ORDINATES or SDO_ELEM_INFO array.

First, a variable of the SDO_GEOMETRY type is defined. In the same statement, where GEOM is declared, its value is also assigned (the value could have been assigned in the body of the anonymous PL*SQL block too).

**Note:** The SDO_ORDINATES or SDO_ELEM_INFO array could have been initialized with more than 999 values.

# Lesson Agenda

- Different ways of loading spatial data
- Loading of spatial data by using SQL*Loader
- Export and import utilities of the Data Pump technology
- Transportable tablespace
- Transactional insert
- Java Shapefile Converter

ORACLE

# Java Shapefile Converter

- Is Oracle's Java-based tool:
  - It is available in Oracle Database 10*g* and 11*g*.
  - The converter is included in the `sdoutl.jar` file that ships with Oracle Database
  - For Oracle Database 10*g*, the following patch is required, which is available on MetaLink:
    - Patch 7636044
- Processes one ESRI shapefile at a time. Processes attributes (`.DBF`) and geometries (`.SHP` and `.SHX`).
- Automatically loads a table with attributes and data
  - Can create a table or add to an existing table
- Loads data in the proper format (rotation, ring ordering)
- Automatically inserts the spatial layer metadata in the `USER_SDO_GEOM_METADATA` view

ORACLE



## Java Shapefile Converter

The Shapefile Converter is Oracle's Java-based tool, available in Oracle Database 10*g* and 11*g*. Oracle Database 10*g* requires patch 7636044, which is available on MetaLink. This tool processes ESRI shapefiles (geometries and attributes) and loads the data into Oracle tables.

The Java Shapefile Converter automatically inserts the metadata about the spatial layer into `USER_SDO_GEOM_METADATA`.

The table that the data is loaded into can be created at load time by the tool. Or, if the table already exists, the spatial data can be appended into the table. It takes care of ring ordering and ensures that exterior polygon elements are followed by all their interior polygon elements.

# Invoking the Java Shapefile Converter

- Specify the classpath after the `-cp` parameter.
- The classpath must include the following files:
  - `%ORACLE_HOME%\jdbc\lib\ojdbc5.jar`
  - `%ORACLE_HOME%\md\jlib\sdoutl.jar`
  - `%ORACLE_HOME%\md\jlib\sdoapi.jar`
- Provide the name of the shapefile without a suffix.

```
java -cp %clpath%
   oracle.spatial.util.SampleShapefileToJGeomFeature
  -h <host_name> -p <port> -s <SID> -u <username>
  -d <password> -f <shapefile> -t <table_name>
  -g <geometry_col> -r <SRID> -o <tolerance>
  -x <x_bounds> -y <y_bounds> -a <append>
  -i <id_col_name> -n <id_start_value>
  -c <commit_interval>
```

ORACLE



## Invoking the Java Shapefile Converter

This slide shows how to invoke the Java Shapefile Converter. Before invoking the tool, you must set the classpath as follows:

```
- Set clpath=.;%ORACLE_HOME%\jdbc\lib\ojdbc5.jar;
    %ORACLE_HOME%\md\jlib\sdoutl.jar;%ORACLE_HOME%\md\jlib\sdoapi.jar
```

The first set of parameters specify the connection information to the database where the table is to be created.

- `<host_name>`: Name of the system that has the Oracle instance to be loaded
- `<port>`: Port number on the host of the listener
- `<SID>`: Name of the Oracle instance to load the data
- `<username>/<password>`: Username and password of the user
- `<shapefile>`: Name of the shapefile to process. Provide the name without any suffix (processes `<shapefile>.shp`, `.shx`, and `.dbf` files).
- `<table_name>`: Parameter that specifies the name of the table to create. If not specified, the table name is the same as the shapefile.
- `<geometry-col>`: Parameter that specifies the geometry column name in the `CREATE TABLE` statement generated by the Shapefile Converter. This column is of the `SDO_GEOMETRY` type.

**Note:** In Oracle Database 10*g*, the equivalent `%ORACLE_HOME%\md\jlib\` folder is `%ORACLE_HOME%\md\lib\`.

## Invoking the Java Shapefile Converter (continued)

- <SRID>: Coordinate system SRID for the data load and USER_SDO_GEOM_METADATA
- <tolerance>: Tolerance value to store in USER_SDO_GEOM_METADATA. For geodetic data, the minimum tolerance must be 0.05 (meters).
- <x_bounds> and <y_bounds>: Bounds of the X and Y dimension to be stored in USER_SDO_GEOM_METADATA. The default for X is –180, 180 and the default for Y is –90, 90.
- <append>: Parameter to add data to an existing table
- <id_column_name>: Column to create and/or hold numeric key value
- <id_start_value>: Number at which to start the numeric key during the load
- <commit_interval>: Parameter to commit every <commit_interval> records

## Running the Java Shapefile Converter: Example

- Input file: `states`
  - Processes `STATES.SHP`, `STATES.SHX`, and `STATES.DBF`
- Output:
  - Table: `geod_states`
  - Column: `geom`
  - `SRID`: `8307`

```
Set clpath=.;%ORACLE_HOME%\jdbc\lib\ojdbc5.jar;
            %ORACLE_HOME%\md\jlib\sdoutl.jar;
            %ORACLE_HOME%\md\jlib\sdoapi.jar
java -cp %clpath%
  oracle.spatial.util.SampleShapefileToJGeomFeature -h
  localhost -p 1521 -s orcl -u student -d student -t
  geod_states -f states -r 8307 -g geom
```

ORACLE



### Running the Java Shapefile Converter: Example

In the slide example, a classpath variable `clpath` is set. Then the sample Java shapefile-to-geometry converter is invoked.

The first set of parameters specifies the connect information to the database where the table is created and the `USER_SDO_GEOM_METADATA` entry is stored.

The next set of parameters specifies that the input shapefile is called `STATES` (no file extension is used; automatically, the `.shp`, `.shx`, and `.dbf` files are used).

The destination table name is `GEOD_STATES`, with a geometry column named `GEOM`. The `SRID` value in `USER_SDO_GEOM_METADATA` and the `SDO_SRID` for each geometry is set to `8307`.

All data loaded by the Java Shapefile Converter is ready to be indexed, and is already in the four-digit, current format.

# Log Output from the Java Shapefile Converter

```
D:\>cd labs\data\shapefile

D:\labs\data\shapefile>set ORACLE_HOME=d:\app\administrator\product\11.1.0\db_1

D:\labs\data\shapefile>set clpath=.;%ORACLE_HOME%\jdbc\lib\ojdbc5.jar;%ORACLE_HO
ME%\md\jlib\sdoutl.jar;%ORACLE_HOME%\md\jlib\sdoapi.jar

D:\labs\data\shapefile>java -cp %clpath% oracle.spatial.util.SampleShapefileToJG
eomFeature -h localhost -p 1521 -s orcl -u student -d student -t geod_states -f
states -r 8307 -g geom
host: localhost
port: 1521
sid: orcl
db_username: student
db_password: student
db_tablename: geod_states
shapefile_name: states
SRID: 8307
db_geometry_column: geom
Connecting to Oracle10g using...
localhost, 1521, orcl, student, student, geod_states, states, null, 8307
Dropping old table...
java.sql.SQLException: ORA-00942: table or view does not exist

Creating new table...
Converting record #10 of 56
Converting record #20 of 56
Converting record #30 of 56
Converting record #40 of 56
Converting record #50 of 56
56 record(s) converted.
Done.
```

ORACLE



## Log Output from the Java Shapefile Converter

The slide shows the log output from running the Java shapefile-to-Oracle Spatial conversion utility. The output shows the information used to connect to the database in which the data will be stored. The input to the converter comes from the `STATES` shapefile and the output table is called `GEOD_STATES`. Additionally, the `SDO_GEOMETRY` column in the `GEOD_STATES` table is called `GEOM`. `SDO_SRID` for each geometry and `SRID` in `USER_SDO_GEOM_METADATA` are set to `8307`. By default, the coordinate system bounds are set to –180 to 180 for longitude, and –90 to 90 for latitude.

# Summary

In this lesson, you should have learned how to:

- Identify the methods for loading spatial data
- Bulk-load spatial data by using SQL*Loader
- Export and import spatial data
- Use the Java Shapefile Converter tool to load an ESRI shapefile into the Oracle Spatial format

ORACLE



## Summary

This lesson discussed different ways of loading spatial data. You also learned how to use the Java Shapefile Converter to load the ESRI shapefile into Oracle Spatial.

# Practice 5: Overview

This practice covers the following topics:

- Loading spatial layers by using the SQL*Loader control and data files
- Importing a `.dmp` file by using the Data Pump import utility
- Using the Java Shapefile Converter to load an ESRI shapefile into the Oracle Spatial format

ORACLE



## Practice 5: Overview

In this practice, first, you load the `GEOD_CITIES` and `GEOD_INTERSTATES` tables by using SQL*Loader. The respective `.ctl` and `.dat` files are located in the `D:\labs\data` folder. Next you import spatial data from the Data Pump import files, `geod_counties.dmp` and `proj_data.dmp`, which are located in the `D:\labs\data` folder. Finally, you use the SampleShapefileToJGeomFeature utility to process a `STATES` shapefile.

## Practice 5

### Loading Spatial Data by Using SQL*Loader

1. The GEOD_CITIES and GEOD_INTERSTATES tables were created in a previous exercise. You now perform the following tasks:
   i. Load the GEOD_CITIES table by using the geod_cities.dat and geod_cities.ctl files.
   ii. Load the GEOD_INTERSTATES table by using the geod_interstates.dat and geod_interstates.ctl files.
   iii. The respective .dat and .ctl files are located in the D:\labs\data folder. Invoke sqlldr to load data in the two tables.

### Loading Spatial Data by Using Data Pump Import

2. Import the GEOD_COUNTIES table from the geod_counties.dmp file. Use Data Pump import. The geod_counties.dmp file is in the D:\labs\data folder.
   **Note:** Before you can use the impdp command to import the GEOD_COUNTIES table, log in using system/oracle as the username/password, and then create a directory object named student_dumpdir using the CREATE DIRECTORY statement. The system user must grant read and write privileges on the student_dumpdir directory to the user student. You can create a log file named geod_counties.log and save it in the D:\labs\data folder, just in case you want to monitor errors, if any.
3. In SQL*Plus, connect again as the student user. Describe the GEOD_COUNTIES table and also check the number of records in the table.
4. Insert metadata for the GEOD_COUNTIES(GEOM) layer into the USER_SDO_GEOM_METADATA view. Commit the record.
   i. Set the tolerance to 0.05.
   ii. Set the SRID to 8307.
   iii. Set the Long axis range to –180 to 180.
   iv. Set the Lat axis range to –90 to 90.
5. Import the following tables from the proj_data.dmp file:
   i. PROJ_CITIES
   ii. PROJ_INTERSTATES
   iii. PROJ_COUNTIES
   iv. PROJ_STATES
   **Note:** Locate the proj_data.dmp file in the D:\labs\data folder. Use the same student_dumpdir directory for the import. Generate a proj_data.log file.
6. Insert metadata for the following layers:
   i. PROJ_CITIES(LOCATION)
   ii. PROJ_INTERSTATES(GEOM)
   iii. PROJ_COUNTIES(GEOM)
   iv. PROJ_STATES(GEOM)
   **Note:** Use the following details for all the tables:
   i. X, -11000000, 4000000, 0.05
   ii. Y, – 80000, 7000000, 0.05
   iii. SRID: 32775

## Practice 5 (continued)

### Using the Java Shapefile Converter Utility

7. Run the Java sample `SampleShapefileToJGeomFeature` utility on the `STATES` shapefile that is provided. Perform the following:
   i. Set the classpath to:
      ```
      .;%ORACLE_HOME%\jdbc\lib\ojdbc5.jar;%ORACLE_HOME%\md\jlib\sdoutl.
        jar;%ORACLE_HOME%\md\jlib\sdoapi.jar.
      ```
      Make sure that `ORACLE_HOME` is set before setting the classpath.
   ii. Run the `SampleShapefileToJGeomFeature` Java-based tool.
   iii. Locate the `STATES` files in the `D:\labs\data\shapefile` folder.
8. Run the `verify_labs.sql` script to check whether all the data is loaded correctly with the correct `SDO_GTYPE`, `SRID`, and number of rows. The `verify_labs.sql` script is located in the `D:\labs\labs` folder.

# 6

# Validating and Debugging Geometries

# Objectives

After completing this lesson, you should be able to do the following:

- Describe the Oracle Spatial validation routines
- Verify the validity of individual geometries or of an entire layer
- Describe geometry debugging routines

ORACLE



## Objectives

In this lesson, you learn about the validation routines available in Oracle Spatial that ensure that the geometries are valid. You are also introduced to debugging routines that help to debug invalid geometries.

# Lesson Agenda

- Validation functions:
  - SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  - SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
- Geometry debugging functions:
  - SDO_UTIL.GETVERTICES
  - SDO_UTIL.RECTIFY_GEOMETRY
  - SDO_UTIL.EXTRACT
- Strategy for geometry validation

ORACLE

# Validating Geometries

- Validation routines ensure that spatial data is valid.
- Geometries are checked to ensure that the defined rules are followed.
- For geometry consistency, the functions check some of the following rules as appropriate to the geometry:
  - Polygons have at least four points, including the point that closes the polygon.
  - Polygons are not self-crossing.
  - No two vertices on a line or polygon are the same.
  - Polygons are oriented correctly.
  - Line strings have at least two points.
  - `SDO_GTYPE` is valid and consistent with the element type.
  - The `SDO_ELEM_INFO` varray contains valid triplet values.

ORACLE



## Validating Geometries

Oracle Spatial provides validation routines that provide detailed information about invalid geometries. These validation routines perform type and geometry consistency checks—for example, they check whether:

- `SDO_GTYPE` is valid
- `element type` is consistent with `SDO_GTYPE`
- The `SDO_ELEM_INFO` varray contains valid triplet values
- Polygons have at least four points, including the point that closes the polygon. The first and last points of a polygon are the same.
- Polygons are not self-crossing and are oriented correctly. Exterior ring boundaries must be oriented counterclockwise, and interior ring boundaries must be oriented clockwise.
- Points on a line are distinct and nonrepeating
- Line strings have at least two points

# Validation Functions

The two validation functions available in the SDO_GEOM package are:

- SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  - Determines whether a geometry is valid
- SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
  - Determines whether all geometries in a layer are valid
- For each invalid geometry, they return the first error encountered
- Also, they return the coordinates where the geometry is invalid

ORACLE



## Validation Functions

Oracle Spatial provides two validation functions that provide detailed information about invalid geometries. The functions are called:

- SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
- SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT

The validation routines provide information about:

- Specific coordinates, where the geometry is invalid
- The specific edge, where the problem occurred
- The specific ring, where the problem occurred

This information makes it easier to debug invalid geometries.

## VALIDATE_GEOMETRY_WITH_CONTEXT Function

```
error := SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
 ( <geometry>, <tolerance> )
```

- VALIDATE_GEOMETRY_WITH_CONTEXT:
  - Checks whether a geometry is valid
  - Returns TRUE if the geometry is valid
  - Returns an Oracle error number and context if invalid (why and where, respectively)
- <geometry>: Is the geometry object to verify
- <tolerance>: Is the tolerance value

ORACLE





### VALIDATE_GEOMETRY_WITH_CONTEXT Function

The SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT function checks for both type consistency and geometry consistency. It accepts an SDO_GEOMETRY object and the tolerance value associated with the geometry as input. The function considers the geometry's tolerance value in determining whether lines touch or whether the points are the same.

The geometry passed in can be selected from a table or it can be a variable or constructor.

If the geometry is valid, the returned value is TRUE.

If the geometry is invalid, the returned value may include:

- A standard Oracle error number that specifies either the reason why the geometry is invalid or FALSE if the reason cannot be determined
- The context that describes where the geometry is invalid. Context information may include:
  - Element number
  - Ring number
  - Edge number

**Note:** This function also has an overloaded format that accepts the DIMINFO array information. If the function format with the DIMINFO array information is used, checking is done to validate that the geometry is within the coordinate system bounds as stored in the DIMINFO field of the USER_SDO_GEOM_METADATA view.

## VALIDATE_GEOMETRY_WITH_CONTEXT: Example

Validating a single geometry at a time:

```
SELECT SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(R.GEOM,
                                               0.05)
FROM RIVERS R
WHERE R.GID = 2;
```

```
SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(GEOM,DIMINFO)
------------------------------------------------------
13341 [Element <1>]
```

Note: The result includes both the error number and the context information about where the error occurred.

```
ORA-13341 = a line geometry has less than two coordinates
```

ORACLE



### VALIDATE_GEOMETRY_WITH_CONTEXT: Example

This is an example of running SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT.

In this example, a single geometry in the RIVERS table is validated.

The query selects the GEOM column from the RIVERS table where the GID column value is 2. The GEOM column is passed into the SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT function along with the tolerance value.

The result includes an error number (13341, which indicates that a line geometry has less than two coordinates) and the context information. The context information in this example specifies that the error occurred in the first element of the geometry.

## VALIDATE_LAYER_WITH_CONTEXT Procedure

```
SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
( <table_name>, <column_name>,
  <result_table> [,<commit_interval>])
```

- Examines a geometry column to determine whether the stored geometries follow the defined rules for geometry objects
- <table_name> and <column_name> identify the layer to validate.
- <result_table>:
  - Validation results are written to this table.
  - This table must be created before calling this procedure.
- <commit_interval> helps track progress. Reports rows processed at the commit interval frequency.

ORACLE



### VALIDATE_LAYER_WITH_CONTEXT Procedure

The SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT procedure validates an entire layer at a time. SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT requires that <result_table> be created before it is called.

Input to the SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT procedure includes:

- <geom_table>: The table name that contains the spatial layer to be validated
- <geom_column>: The column name of the SDO_GEOMETRY type that contains the spatial layer
- <result_table>: The name of the table that the results of the function are written to. The table must be created before running the function. A row is added to this table for each invalid geometry. The next slide has more information about <result_table>.
- <commit_interval>: This optional parameter specifies the frequency to commit results to the <result_table>. The commit interval can be used to determine how much of the layer has been validated. If no <commit_interval> is specified, the default is to commit after the entire layer has been validated. You can query <result_table> to monitor how many rows have been processed.

# Structure of the Results Table

- The results table holds the validation results.
- It must be created before using the procedure.
- Column names must be defined in data type order.

```
CREATE TABLE validation_results (
  sdo_rowid    ROWID,
  status       VARCHAR2(2000)
);
```

ORACLE





## Structure of the Results Table

The table that holds the results of the `SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT` procedure must exist before the procedure is run. The table name and column names are user defined. `SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT` requires that the result table contain columns of the `ROWID` and `VARCHAR2(2000)` types, in that respective order. The columns must be created in that order.

In each row for an invalid geometry, the `SDO_ROWID` column contains the `ROWID` value of the row containing the invalid geometry, and the `RESULT` column contains an Oracle error message number and the context of the error (the coordinate, edge, or ring that causes the geometry to be invalid).

## VALIDATE_LAYER_WITH_CONTEXT: Example

```
BEGIN
  SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
      ('RIVERS', 'GEOM', 'VALIDATION_RESULTS');
END;
/
SELECT * FROM validation_results;
```

The VALIDATION_RESULTS table contains:

```
SDO_ROWID            STATUS
---------------      ------------------------------------
                     Rows Processed <752>
AAADCsAABAAAPUpAAA   13341 [Element <1>]
AAADCsAABAAAPUpAAB   13356 [Element <1>] [Coordinate <2>]
AAADCsAABAAAPUpAAC   13356 [Element <1>] [Coordinate <4>]
AAADCsAABAAAPUpAAD   13356 [Element <1>] [Coordinate <3>]
AAADCsAABAAAPUpAAE   13356 [Element <1>] [Coordinate <4>]
```

```
ORA-13341 = a line geometry has less than two coordinates
ORA-13356 = adjacent points in a geometry are redundant
```

ORACLE



### VALIDATE_LAYER_WITH_CONTEXT: Example

The slide example validates all the geometries in the GEOM column of the RIVERS table.

The results of the procedure are stored in the VALIDATION_RESULTS table. Because a value was not specified for <commit-interval>, the procedure runs against the entire layer before committing the results to the VALIDATION_RESULTS table.

The SELECT statement after the SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT procedure call shows the results stored in the VALIDATION_RESULTS table. The data in the SDO_ROWID column is the ROWID associated with the geometry in the original table, and the text in the STATUS column is the error code and context information associated with that geometry.

In this example, the geometry with GID=2 in the RIVERS table has an error code of 13341, which means that the line geometry has less than two coordinates, and the context specifies that the error occurred in Element 1. All the other geometries have an error code of 13356, which means that adjacent points in the line string are redundant. For this error, the context information reports the first redundant coordinate.

A subsequent slide shows how to look at the coordinates where the error occurred. For a complete description of error codes, refer to the *Oracle Database Error Messages* manual.

# Lesson Agenda

- Validation functions:
  - SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  - SDO_GEOM.VALIDATE_LAYER_ _WITH_CONTEXT
- Geometry debugging functions:
  - SDO_UTIL.GETVERTICES
  - SDO_UTIL.RECTIFY_GEOMETRY
  - SDO_UTIL.EXTRACT
- Strategy for geometry validation

ORACLE

# Geometry Debugging Functions

Functions to help debug invalid geometries:

- SDO_UTIL.GETVERTICES
- SDO_UTIL.RECTIFY_GEOMETRY
- SDO_UTIL.EXTRACT

ORACLE



## Geometry Debugging Routines

You can use the following functions to debug invalid geometries:

- SDO_UTIL.GETVERTICES: Returns formatted coordinates
- SDO_UTIL.RECTIFY_GEOMETRY: Fixes certain problems with the input geometry and returns a valid geometry
- SDO_UTIL.EXTRACT: Returns a single element from a geometry, or optionally returns a single ring of a polygon element

# SDO_UTIL.GETVERTICES Function

```
VERTEX_SET_TYPE := SDO_UTIL.GETVERTICES(geometry);
```

- SDO_UTIL.GETVERTICES:
  - Returns the coordinates of the vertices of the input geometry
  - Is useful in finding a vertex that is causing a geometry to be invalid
  - Returns MDSYS.VERTEX_SET_TYPE
    - This is defined as TABLE of MDSYS.VERTEX_TYPE.
    - VERTEX_TYPE is an object that contains x, y, z, w, and id attributes of the NUMBER data type.
- <geometry>: Is the SDO_GEOMETRY object whose vertices are to be returned

ORACLE



## SDO_UTIL.GETVERTICES Function

The input to the SDO_UTIL.GETVERTICES function is a geometry whose vertices are to be extracted.

The result of the function is a nested table of MDSYS.VERTEX_TYPE that can have up to four ordinates for each row.

- A two-dimensional geometry returns values in the x and y fields.
- A three-dimensional geometry returns values in the x, y, and z fields.
- A four-dimensional object returns values in all four fields. w is for measure value.

Using this function, you can isolate problems identified by the SDO_UTIL.VALIDATE_GEOMETRY_WITH_CONTEXT function and the SDO_UTIL.VALIDATE_LAYER_WITH_CONTEXT procedure. This function can be useful in finding a vertex that causes a geometry to be invalid. You can view the vertices in the tabular format.

## SDO_UTIL.GETVERTICES: Example

- Validation returns redundant adjacent points error (13356).

```
SDO_ROWID          STATUS
---------------    -----------------------------------
AAADCsAABAAAPUpAAD 13356 [Element <1>] [Coordinate <3>]
```

- Use SDO_UTIL.GETVERTICES to see redundancy.

```
SELECT rownum, gv.x, gv.y
FROM rivers r,
     TABLE(sdo_util.getvertices(r.geom)) gv
WHERE rowid = 'AAADCsAABAAAPUpAAD'
  AND rownum <= 4;
```

```
ROWNUM          X            Y
--------- ----------   ----------
        1 -71.017892   42.2910630
        2 -70.974900   42.3042252
        3 -70.957062   42.3360129
        4 -70.957062   42.3360129
```

ORACLE



### SDO_UTIL.GETVERTICES: Example

The slide shows how the SDO_UTIL.GETVERTICES function can help you identify the vertices that are causing the geometry to be invalid. The geometry at the given row ID returns an Oracle error number of 13356, which states that adjacent vertices in the geometry are redundant. Further, the SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT procedure also reports that the error occurred at Element 1, Coordinate 3.

The SELECT statement reports the duplicate coordinate identified in the validation routine.

Note that SDO_UTIL.GETVERTICES is specified in the FROM clause. The WHERE clause looks at the ROWID of the geometry reported in the validation_results table (in the slide example, identified by ROWID AAADCsAABAAAPUpAAD), and also specifies the terminating criteria of how many vertices to display (display four vertices). The error was reported at Coordinate 3, which is the first of the two redundant points. Specifying rownum<= 4 displays both the redundant coordinates.

## SDO_UTIL.RECTIFY_GEOMETRY Function

- This function fixes the following problems with the input geometry and returns a valid geometry:
  - Duplicate vertices
  - Self-crossing polygons
  - Incorrect orientation of the outer and inner rings in a polygon
- For any other issue, the function raises an exception.

```
SDO_GEOMETRY := SDO_UTIL.RECTIFY_GEOMETRY(
                                    <geometry>,<tolerance>)
```

ORACLE



### SDO_UTIL.RECTIFY_GEOMETRY Function

The SDO_UTIL.RECTIFY_GEOMETRY function can be used to fix invalid geometries. This function checks for the following problems that can make a geometry invalid, and fixes the problems in the returned geometry:

- Duplicate or redundant vertices
- A polygon boundary intersecting itself
- Incorrect orientation of exterior or interior rings (or both) of a polygon

If the input geometry has any other problem that makes it invalid, the function raises an exception. If the input geometry is valid, the function returns a geometry that is identical to the input geometry.

## SDO_UTIL.RECTIFY_GEOMETRY Function: Example

```
SELECT SDO_UTIL.RECTIFY_GEOMETRY(
  SDO_GEOMETRY(2002, 8307, NULL,
    SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(
      -71.017892, 42.2910630, -70.974900, 42.3042252,
      -70.957062, 42.3360129, -70.957062, 42.3360129,
      -70.974857, 42.3678167)), 0.5)
FROM dual;
```

The query returns the geometry without the duplicate vertex.

```
SDO_GEOMETRY(2002, 8307, NULL,
  SDO_ELEM_INFO_ARRAY(1, 2, 1),
  SDO_ORDINATE_ARRAY(
    -71.017892, 42.2910630, -70.974900, 42.3042252,
    -70.957062, 42.3360129, -70.974857, 42.3678167))
```

ORACLE



### SDO_UTIL.RECTIFY_GEOMETRY Function: Example

SDO_UTIL.RECTIFY_GEOMETRY returns a new geometry by removing adjacent vertices in the input geometry that are closer together than a specified tolerance.

# SDO_UTIL.EXTRACT Function

```
SDO_GEOMETRY := SDO_UTIL.EXTRACT
   ( <geometry>, <element> [, <ring>])
```

- SDO_UTIL.EXTRACT:
  - Extracts an element or subelement from a geometry
  - Applies only to two-dimensional geometries
- <geometry>: Is the SDO_GEOMETRY object to extract from
- <element>: Is the number of the element to extract (starts from 1)
- [<ring>]: Is valid for a polygon with zero or more holes
  - Ring1 represents the outer ring, Ring2 represents the first inner ring, and so on.

ORACLE



## SDO_UTIL.EXTRACT Function

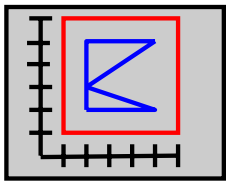
The SDO_UTIL.EXTRACT function accepts as input the geometry from which to extract an element or subelement. The function returns a geometry that is the element (or subelement) specified within the input geometry to extract. This function can be used to extract the invalid element from a geometry.

Geometries with the rightmost digit in SDO_GTYPES ending in 1, 2, or 3 have only one element. Geometries with the rightmost digit in SDO_GTYPE ending in 4, 5, 6, or 7 can have one or more elements. Individual elements within a geometry can be extracted with the SDO_UTIL.EXTRACT function.

Subelements can be extracted from geometries with the leftmost digit in SDO_GTYPE ending in 3 (polygon), 4 (collection), 5 (multipoint), or 6 (multipolygon).

Use the validation functions discussed previously to identify which element in a geometry has a problem, and examine the element more closely by using the SDO_UTIL.EXTRACT function to isolate it.

## SDO_UTIL.EXTRACT: Example



```
SELECT sdo_geom.validate_geometry_with_context(
         sdo_geometry(2003, 32775, NULL,
           sdo_elem_info_array(1,1003,1, 11,2003,1),
           sdo_ordinate_array(1,1,6,1,6,6,1,6,1,1,
                              2,2,2,5,5,5,2,3,5,2,2,2)),
       0.5) V_W_C
FROM dual;
```

```
V_W_C
-----------------------------------------------------
13349 [Element <1>] [Ring <2>][Edge <3>][Edge <1>]
```

```
SELECT sdo_util.extract(sdo_geometry (2003, 32775, NULL,
       sdo_elem_info_array(1,1003,1, 11,2003,1),
       sdo_ordinate_array(1,1,6,1,6,6,1,6,1,1,
                          2,2,2,5,5,5,2,3,5,2,2,2)),
       1,2) EXTR_GEOM
FROM dual;
```

```
EXTR_GEOM
-----------------------------------------------------
SDO_GEOMETRY(2003, 32775, NULL,
  SDO_ELEM_INFO_ARRAY(1, 1003, 1),
  SDO_ORDINATE_ARRAY(2, 2, 5, 2, 2, 3, 5, 5, 2, 5, 2, 2))
```

ORACLE



### SDO_UTIL.EXTRACT: Example

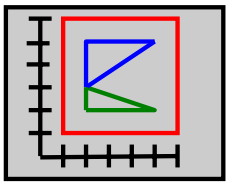
This example shows how the SDO_UTIL.EXTRACT function can be used to extract the invalid portion of a geometry. The validation function returned an Oracle error number of 13349. This error is returned when a polygon boundary crosses itself. Further, the SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT procedure also reported the error was at Element 1, Ring 2, and that the edges that self-crossed were Edge 3 and Edge 1.

The geometry in this example is small, and it would not be difficult to find the element or subelement that is invalid. But many geometries have several elements and subelements. For those geometries, it is more difficult to identify the element that is making a geometry invalid. The SDO_UTIL.EXTRACT function is used to extract and isolate the broken subelement.

The parameters specified for the SDO_UTIL.EXTRACT function match the context information returned from SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT.
1 matches [Element <1>], and 2 matches [Ring <2>].

**Note:** If the subelement is a hole (or void), its rotation is reversed when it is extracted. This means that the extracted inner ring becomes an outer ring.

# Using the SDO_UTIL.RECTIFY_GEOMETRY Function



```
SELECT sdo_util.rectify_geometry(
       sdo_geometry (2003, 32775, NULL,
          sdo_elem_info_array(1,1003,1, 11,2003,1),
          sdo_ordinate_array(1,1,6,1,6,6,1,6,1,1,
                             2,2,2,5,5,5,2,3,5,2,2,2)),
       0.5) EXTR_GEOM
FROM dual;
```

```
SDO_GEOMETRY(2003, 32775, NULL,
  SDO_ELEM_INFO_ARRAY(1, 1003, 1, 11, 2003, 1, 19, 2003, 1),
  SDO_ORDINATE_ARRAY(1,6, 1,1, 6,1, 6,6, 1,6,
                     2,3, 5,2, 2,2, 2,3,
                     2,5, 5,5, 2,3, 2,5))
```

ORACLE

# Lesson Agenda

- Validation functions:
  - SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT
  - SDO_GEOM.VALIDATE_LAYER_ _WITH_CONTEXT
- Geometry debugging functions:
  - SDO_UTIL.GETVERTICES
  - SDO_UTIL.RECTIFY_GEOMETRY
  - SDO_UTIL.EXTRACT
- Strategy for geometry validation

ORACLE

## Strategy for Geometry Validation

1. Create a `validation_results` table (as specified in the previous slides), and then run the `VALIDATE_LAYER_WITH_CONTEXT` function.
2. For each type of error, create a table and isolate the invalid geometries.
3. Remove the invalid geometries from your spatial layer.
4. Run the `SDO_UTIL.RECTIFY_GEOMETRY` function on the new table containing the invalid geometries.
5. Validate the geometries again.
6. Insert the fixed geometries back into the original table.
7. Repeat this process for each type of validation error.

ORACLE



### Strategy for Geometry Validation

The slide lists the strategy that you can follow to validate your spatial data.

# Strategy for Geometry Validation: Example

Run the VALIDATE_LAYER_WITH_CONTEXT function.

```
BEGIN
  SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT
      ('ROADS', 'GEOM', 'VALIDATION_RESULTS');
END;
/
SELECT * FROM validation_results;
```

```
SDO_ROWID            STATUS
-----------------------------------------------------
                     Rows Processed <752>
AAADCsAABAAAPUpAAA   13341 [Element <1>]
AAADCsAABAAAPUpAAB   13356 [Element <1>] [Coordinate <2>]
AAADCsAABAAAPUpAAC   13356 [Element <1>] [Coordinate <4>]
AAADCsAABAAAPUpAAD   13356 [Element <1>] [Coordinate <3>]
AAADCsAABAAAPUpAAE   13356 [Element <1>] [Coordinate <4>]
```

```
ORA-13341 = a line geometry has less than two coordinates
ORA-13356 = adjacent points in a geometry are redundant
```

ORACLE



## Strategy for Geometry Validation: Example

Assuming that you already created the validation_results table, run the VALIDATE_LAYER_WITH_CONTEXT function.

# Strategy for Geometry Validation: Example

Repeat the following for each type of error:

- Isolate the invalid geometries.

```
CREATE TABLE roads_13356 AS
  SELECT *
  FROM roads
  WHERE rowid in (SELECT sdo_rowid
                  FROM validation_results
                  WHERE substr(STATUS, 1, 5) = 13356);
```

- Remove the invalid geometries from your spatial layer.

```
DELETE FROM roads
WHERE rowid in (SELECT sdo_rowid
                FROM validation_results
                WHERE substr(STATUS, 1, 5) = 13356);
```

ORACLE



## Strategy for Geometry Validation: Example (continued)

Create tables for each type of error and isolate the invalid geometries. Delete the invalid geometries from the original table.

## Strategy for Geometry Validation: Example

- SDO_UTIL.RECTIFY_GEOMETRY fixes geometries.

```
UPDATE roads_13356
SET geom = SELECT sdo_util.rectify_geometry
                            (geom, tolerance);
```

- Run validation on ROADS_13356.
- Insert the fixed geometries back into the ROADS table.
- Repeat this process for each type of validation error.

ORACLE



### Strategy for Geometry Validation: Example (continued)

Run SDO_UTIL.RECTIFY_GEOMETRY on the new table to debug the invalid geometries. Run the validation routine again on the new table to determine whether the geometries are valid. Insert the fixed geometries back into the original table.

# Modifying Geometries in PL/SQL

- PL/SQL collection (`VARRAY`) methods:
  - `.COUNT`
  - `.EXTEND`
  - `.TRIM`
  - Others
- Reference: *PL/SQL User's Guide and Reference* under *Using PL/SQL Collections and Records*

ORACLE



## Modifying Geometries in PL/SQL

The *PL/SQL User's Guide and Reference* manual refers to `VARRAYS` as collections. The manual has a section that defines methods on PL/SQL collections (`or VARRAYS`). These methods are very useful to modify the `SDO_ELEM_INFO` and `SDO_ORDINATES` fields of the `SDO_GEOMETRY` object.

PL/SQL collection (`VARRAY`) methods include:

- `.COUNT`: Returns the number of array entries in the varray
- `.EXTEND`: Adds empty array entries to the varray
- `.TRIM`: Removes a specified amount of array entries from the end of a `VARRAY`
- Others

See the *PL/SQL User's Guide and Reference* manual, under *Using PL/SQL Collections and Records*.

Some examples are located in the `D:\labs\demo` directory for this course:

- `create_geom.sql`
- `modify_geom.sql`

# Summary

In this lesson, you should have learned how to:

- Describe the Oracle Spatial validation routines
- Verify the validity of individual geometries or of an entire layer
- Describe geometry debugging routines

ORACLE



## Summary

In this lesson, you learned about the validation routines available in Oracle Spatial that ensure that the geometries are valid. You also learned about the debugging routines that help debug invalid geometries.

# Practice 6: Overview

This practice covers the validation of spatial objects by using the validation functions.

ORACLE



## Practice 6: Overview

In this practice, you insert some invalid geometries, and then use the validation functions to validate them. You also use the debugging functions to correct the geometries.

## Practice 6

1. Before you use the validation and debugging functions, insert a few invalid geometries in the `geom_data` table. Run the `lab_06_01.sql` script to insert invalid geometries in the `geom_data` table. The `lab_06_01.sql` script is in the `D:\labs\labs` folder.
2. Validate the geometry with Object_id as `10`.
3. Create a `val_results` table with the following structure:

| Column name | Data type |
|---|---|
| `sdo_rowid` | `ROWID` |
| `result` | `VARCHAR2(1000)` |

4. Validate the `geom_data(shape)` spatial layer and review the validation results.
5. Write a query to return the vertices of the geometry object with Object_id as `13`.
6. Remove the redundant vertices from the geometry object with Object_id as `13`.
7. Debug the geometry object with Object_id `11`. Use the `RECTIFY_GEOMETRY` function. First, verify in the `val_results` table and find out what is wrong with the geometry.
8. From the geometry object identified by Object_id as `8`, extract the first element.

7

# Using the Oracle Application Server MapViewer

# Objectives

After completing this lesson, you should be able to do the following:

- Describe MapViewer
- Describe the components of MapViewer
- Install and configure Oracle Application Server Containers for J2EE (OC4J) with the MapViewer Quickstart kit
- Render query results with the `JView.jsp` demo
- Edit the `MapViewerConfig.xml` file to configure a data source
- Configure the logging level reported by MapViewer

ORACLE



## Objectives

This lesson introduces you to the concepts associated with Oracle Application Server MapViewer.

# Lesson Agenda

- Introduction to MapViewer
- Architecture of Oracle Application Server MapViewer
- Installation of Oracle Application Server MapViewer
  - Adding an OC4J data source
- MapViewer demos
- Debugging

# MapViewer

- Is a map-rendering engine that you can use to easily publish spatial data to the Web
- Is not part of Oracle Spatial or Oracle Locator
  - Is integrated with Oracle Spatial and Oracle Locator
- Is a feature of Oracle Application Server and requires an Oracle Application Server license
- Provides XML, AJAX JavaScript, Java, JSP tag library, and Open Geospatial Consortium (OGC) Web Map Service (WMS) APIs
- Is a pure Java servlet
- Can render maps from Oracle 8.1.7 databases and later
- Includes Map Builder, a tool to administer map metadata such as style, theme, and map definitions

ORACLE



## MapViewer

MapViewer is a feature introduced in Oracle9*i*AS Release 2, with ongoing availability in Oracle Application Server. MapViewer is a map-rendering engine that you can use to easily publish spatial data to the Web. MapViewer is not part of Oracle Spatial or Oracle Locator, but it is integrated with both. Publishing maps to the Web with MapViewer does require licensing Oracle Application Server.

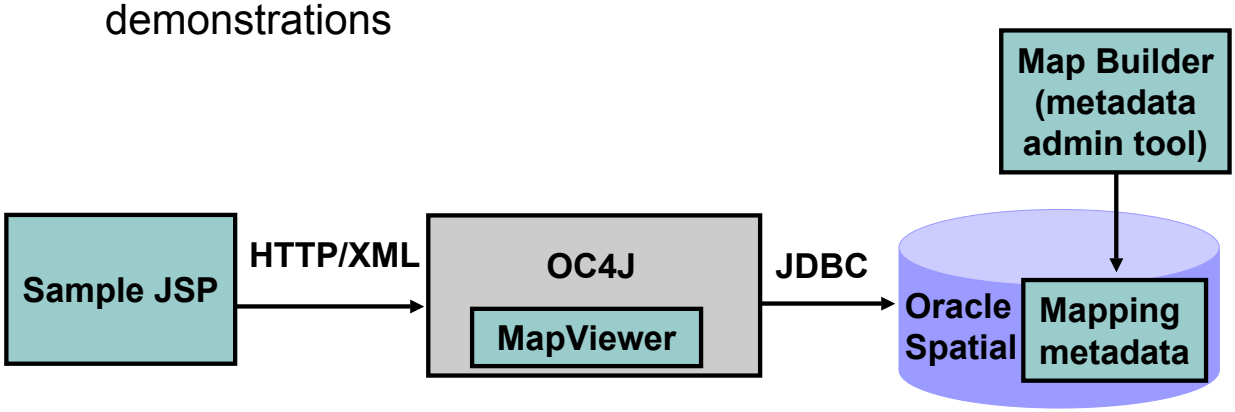
MapViewer has a simple XML API, which is accessible through HTTP or any programming interface that can submit an XML request. Additionally, there are AJAX JavaScript, Java, JSP tag library, and Open Geospatial Consortium Web Mapping Service (OGC WMS) APIs.

MapViewer is a pure Java servlet, and can render map data stored in Oracle 8.1.7 databases and later. MapViewer includes Map Builder, an administrative tool to help populate map metadata required to render maps. The Map Builder tool is a stand-alone Java application.

## MapViewer Installation and Configuration

A MapViewer installation includes the following components:

- An enterprise archive (`mapviewer.ear`) file
- The Map Builder administration tool
- MapViewer metadata views
- Several JavaServer Pages (JSP) and AJAX JavaScript demonstrations




ORACLE



### MapViewer Installation and Configuration

A MapViewer installation includes the following components:

- The `MapViewer.ear` (enterprise archive) file. The latest version of this file can be downloaded from Oracle Technology Network (OTN):
  http://www.oracle.com/technology/software/products/mapviewer
- Map metadata views that are automatically created during an Oracle database installation
- The Map Builder administration tool, which is used to manage map metadata
- Several JavaServer Page (JSP) and AJAX JavaScript demonstrations

# MapViewer and Oracle Application Server

- MapViewer is available with the licensed Oracle Application Server.
- MapViewer is a feature included in every edition of Oracle Application Server:
  - Java edition
  - Standard edition
  - Enterprise edition
- If you own Oracle Application Server, you own MapViewer (no extra cost for MapViewer).

ORACLE



## MapViewer and Oracle Application Server

MapViewer requires Oracle Application Server Containers for J2EE (OC4J), which is part of Oracle Application Server. MapViewer installation and registration with Oracle Application Server are discussed in detail in the:

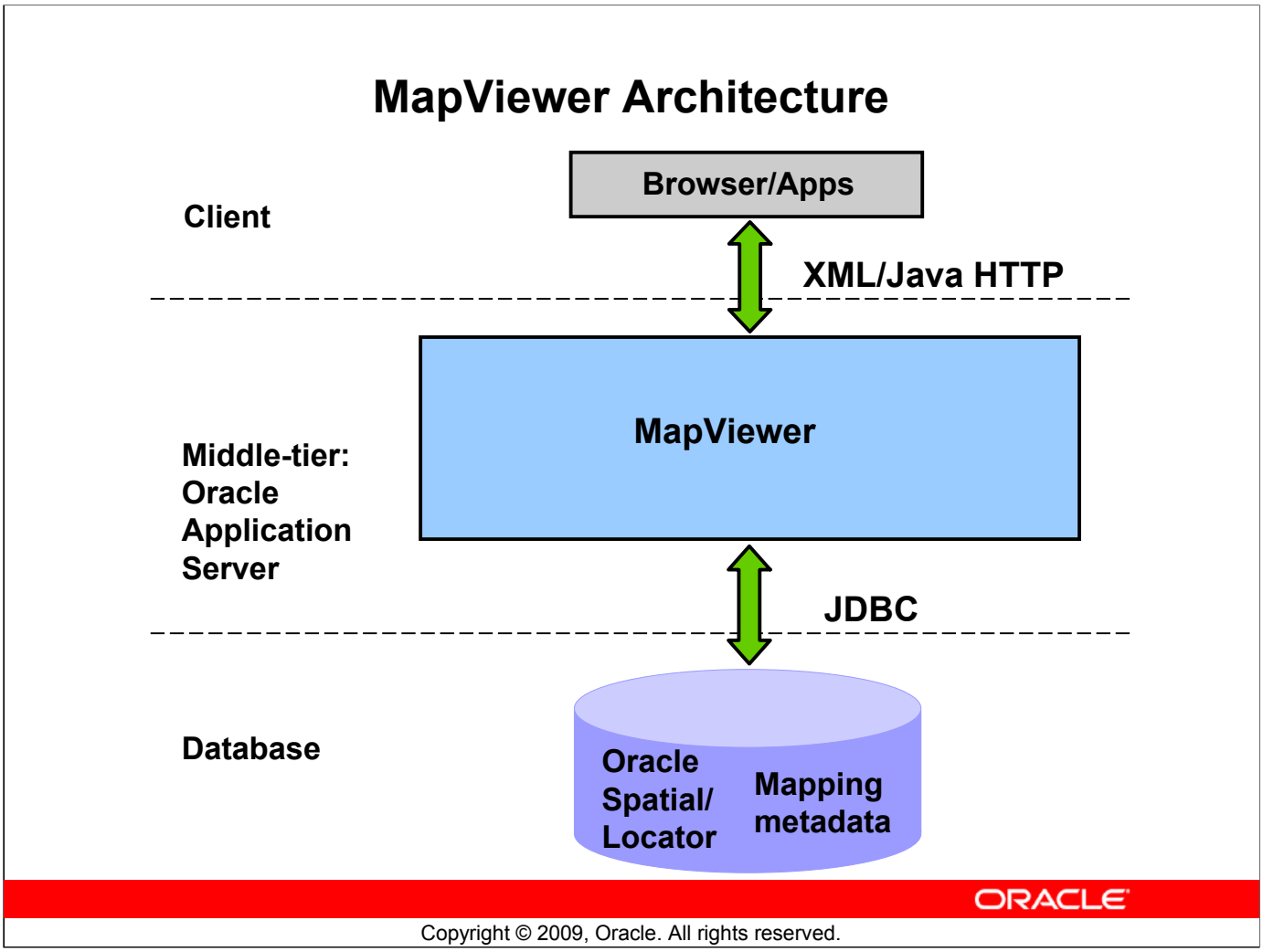
- *Oracle Application Server MapViewer User's Guide*
- MapViewer FAQ located at:
  http://www.oracle.com/technology/software/products/mapviewer/index.html

When you deploy an application, it is recommended that you use features in Oracle Application Server that address high availability, load balancing, and so on.

The *Oracle Application Server MapViewer User's Guide* has a section that discusses deploying MapViewer on a multiprocess OC4J instance.

# Lesson Agenda

- Introduction to MapViewer
- Architecture of Oracle Application Server MapViewer
- Installation of Oracle Application Server MapViewer
  - Adding an OC4J data source
- MapViewer demos
- Debugging




## MapViewer Architecture

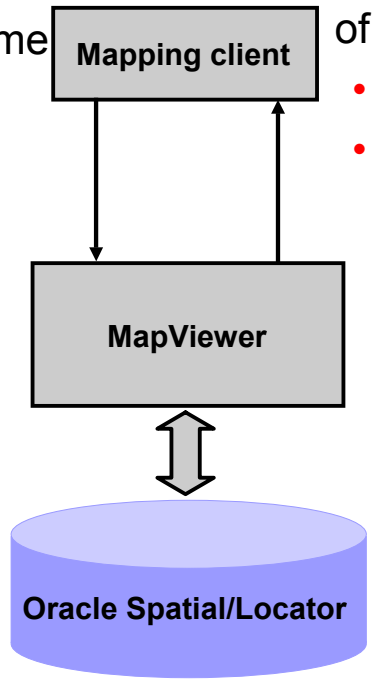
Client applications connect to MapViewer and send XML mapping requests via the HTTP communications protocol. Requests include information about which database to query, and also information about the map to be drawn.

MapViewer connects to the database by using Java Database Connectivity (JDBC), and either dynamically determines information about the map to be drawn or fetches information about the map to be drawn from the map metadata tables. It then selects the spatial data to be mapped by performing spatial queries. MapViewer then renders the map. The response to the map request sent to the client for display is a streamed map or a pointer to the map along with the bounds of the map.

## MapViewer Query

A map request consists of:

- The base map name
- Center of the map
- Width and height of the map
- Optional tags:
  - Map name
  - `jdbc_query`
  - Others




A map response consists of one of the following:

- A streamed map image
- A URL to the map image along with the map MBR

ORACLE

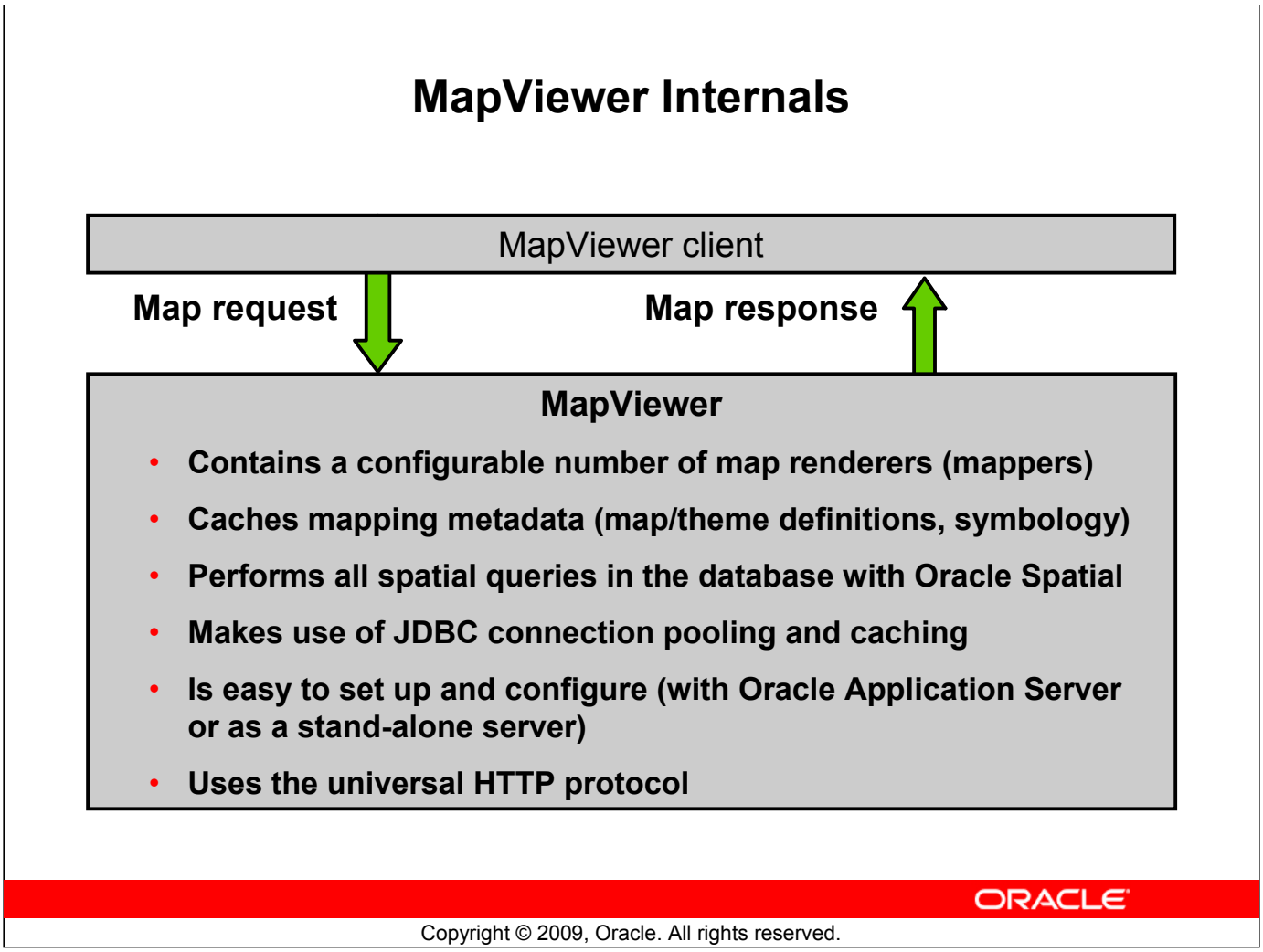


### MapViewer Query

A client sends an Extensible Markup Language (XML) request to MapViewer that includes the base map name as stored in the map metadata, the center and size of the map to be displayed, and any optional tags to be displayed in the map (such as a map title, `<jdbc_query>` elements, legends, and so on).

`<jdbc_query>` elements enable you to submit any SQL statement in Oracle that returns `SDO_GEOMETRY` objects, and display the results on a base map generated by MapViewer. The base map generated by MapViewer is composed of rectangular viewport queries against a predefined set of `SDO_GEOMETRY` columns. `<jdbc_query>` elements are discussed later in this lesson.

MapViewer queries map metadata views via JDBC to determine which layers are associated with the map that it is about to generate. MapViewer queries the spatial data layers, renders the data, performs the JDBC element queries, and then returns either a streamed map or a pointer to the map, along with the minimum bounding rectangle (MBR) associated with the map.




## MapViewer Internals

XML map requests come to the MapViewer servlet from a client application via HTTP.

When MapViewer is configured, the number of "mappers" can be specified. Mappers are rendering processes that process one map request at a time. MapViewer manages the map request queue, and sends requests to available mappers.

Mappers use JDBC connection pooling. As mappers receive requests, they perform JDBC queries to fetch data from the Oracle database, and build maps. Each mapper component uses a cache to reduce query traffic to the server for map metadata.

The mapper metadata cache can be flushed through the MapViewer Admin JSP (discussed later in this lesson).

# MapViewer and Oracle Application Server

- Installing MapViewer after an Oracle Application Server installation (recommended for deployment) is discussed in:
  - *Oracle Application Server MapViewer User's Guide*
  - MapViewer FAQ
- OC4J has its own Web listener, and OC4J can be installed independently (or stand-alone) from Oracle Application Server.
- Installing OC4J stand alone still requires that Oracle Application Server be licensed.
- The MapViewer Quickstart kit is a quick way to get started. It contains a stand-alone OC4J container with MapViewer predeployed.

ORACLE



## MapViewer and Oracle Application Server

For development purposes, and for the purpose of this course, you can use a stand-alone version of OC4J.

OC4J has its own Web listener, and OC4J can be installed independently (or as stand-alone) from Oracle Application Server. When installed, OC4J occupies about 150 MB of disk space. Installing OC4J stand-alone still requires licensing Oracle Application Server.

From the OTN MapViewer page, you can download the MapViewer Quickstart kit. The MapViewer Quickstart kit is a stand-alone OC4J instance with MapViewer predeployed. This is a quick and easy way to install and run MapViewer.

# Lesson Agenda

- Introduction to MapViewer
- Architecture of Oracle Application Server MapViewer
- Installation of Oracle Application Server MapViewer
  - Adding an OC4J data source
- MapViewer demos
- Debugging

# Stand-Alone OC4J Installation with the MapViewer Quickstart Kit

- Requires at least Java 1.5 JDK
  - It is included when Oracle 11*g* is installed: `%ORACLE_HOME%\jdk\bin`
  - Make sure that Java 1.5 JDK or later is in your `PATH` environment variable.
  - To verify, enter `java -version`.
- MapViewer Quickstart kit:
  - Contains stand-alone OC4J, predeployed with MapViewer
  - Can be downloaded from the MapViewer page on OTN:
    - http://www.oracle.com/technology/products/mapviewer
    - At the right side of the page, click Software.

ORACLE



## Stand-Alone OC4J Installation with the MapViewer Quickstart Kit

OC4J requires a Java Development Kit (JDK)—at least, version 1.5—in which to execute. To check the Java version installed, enter `java -version` at the operating system prompt. Make sure that the JDK's `bin` directory is included in your `PATH` environment variable.

Also, to ensure that you have a JDK in your path, and not just a Java Runtime Environment (JRE), at the operating system prompt, enter `javac` (this is the java compiler). If `javac` is not found, either you do not have a JDK installed, or you do have a JDK installed, but the JDK's bin directory is not listed in your `PATH` environment variable.

**Note:** If the Java version is older than 1.5, visit http://java.sun.com/javase/downloads/index.jsp to download the latest Java SE Development Kit (JDK). Install JDK and make sure that you set the `PATH` variable to include the JDK's `bin` directory.

## Installing the MapViewer Quickstart Kit

- Unzip the `mv10131_qs.zip` file and the `mv10131_qs` directory is created.
- The `mv10131_qs` directory contains:
  - The `OC4J` directory: OC4J, preconfigured to deploy MapViewer
  - `readme.txt`: Description of how to start and access MapViewer
  - `start.bat`: Script to start MapViewer on Windows
  - `start.sh`: Script to start MapViewer on UNIX
- Edit `start.bat` or `start.sh`:
  - Use the correct path to the location of `mv10131qs\oc4j\j2ee\home`.
  - Use the correct path to the location of `java`.
  - Adjust the Java heap size if necessary (for example, `-Xmx384m`).
- Run `start.bat` if on Windows (or `start.sh` if on UNIX).
  - Enter the password for the `OC4JADMIN` user when prompted.

ORACLE



### Installing the MapViewer Quickstart Kit

To install the MapViewer Quickstart kit, unzip the `mv10131_qs.zip` file. `mv10131_qs` is created. The directory contains:

- The OC4J directory. This is a complete stand-alone instance of OC4J.
- A `readme.txt` file that describes how to start and access MapViewer
- `start.bat`, a Windows-specific script that can be run to start OC4J and MapViewer
- `start.sh`, a UNIX-specific script that can be run to start OC4J and MapViewer

Edit either `start.bat` or `start.sh` depending on which operating system you work on. In the file:

- Make sure that the path to `mv10131qs\oc4j\j2ee\home` is correct
- Make sure that the path to the `java` executable is correct
- Adjust the Java heap size if necessary. If your typical map contains many geometries, you may need to increase your Java heap size.

To start OC4J and MapViewer, execute either `start.bat` or `start.sh` depending on your operating system. A read-only console window opens. The console is where MapViewer sends status and error message information.

# Installing the MapViewer Quickstart Kit

- In the console window where OC4J was started, you see:

```
INFO [oracle.lbs.mapserver.oms] *** Oracle MapViewer
started. ***
```

- `%OC4J_HOME%` is located at `mv10131_qs\oc4j\j2ee\home`.
- The QuickStart kit contains `%OC4J_HOME%\applications\mapviewer.ear` (an enterprise archive file).
- Newer versions of `mapviewer.ear` may be located on the MapViewer page on OTN:
  - http://www.oracle.com/technology/products/mapviewer
  - On the right side of the page, click Software.

ORACLE



## Installing the MapViewer Quickstart Kit (continued)

If MapViewer starts successfully, you see the following message in the OC4J console:

```
INFO [oracle.lbs.mapserver.oms] *** Oracle MapViewer started. ***
```

The OC4J home is an important reference directory. Its location is `mv10131_qs\oc4j\j2ee\home`.

The MapViewer Quickstart OC4J home contains the `mapviewer.ear` file (an enterprise archive file).

Its location is `%OC4J_HOME%\applications\mapviewer.ear`.

Newer versions of the `mapviewer.ear` file may be available on the MapViewer home page on OTN, located at http://www.oracle.com/technology/products/mapviewer.

# Installing the MapViewer Quickstart Kit

- The first time you run `start.bat` (or `start.sh`), MapViewer deploys the `mapviewer.ear` file.
- `mapviewer.ear` expands into a directory structure. The following are the main subdirectories:

```
%OC4J_HOME%/applications/mapviewer/
                                    sql/
                                    web/
                                        fsmc/
                                        WEB-INF/
                                                lib/
                                                conf/
                                                log/
                                                mapcache/
                                                classes/
                                                admin/
```

## Updating `mapviewer.ear` with a Newer Version

- Copy the newer version of `mapviewer.ear` (for example, from the MapViewer page on OTN) to the `%OC4J_HOME%/applications/` directory.
- Delete or rename the old `%OC4J_HOME%/applications/mapviewer` directory.
  - Renaming saves the `MapViewerConfig.xml` file and any customizations.
- Restart OC4J by running `start.bat` if on Windows or `start.sh` if on UNIX.
- The new `mapviewer.ear` file expands as `%OC4J_HOME%/applications/mapviewer`.

ORACLE



### Updating `mapviewer.ear` with a Newer Version

It is very easy to update your MapViewer instance with a newer version.

- First, copy the newer version of `mapviewer.ear` to the `%OC4J_HOME%/applications/` directory.
- Either delete or rename the existing `%OC4J_HOME%/applications/mapviewer` directory. It has to be deleted or removed, so the new `mapviewer.ear` file can expand in its place.
- Restarting OC4J expands the new `mapviewer.ear` file as `%OC4J_HOME%/applications/mapviewer`.

# Opening the MapViewer Home Page

**http://localhost:8888/mapviewer/**


- First, set up an OC4J data source.
- An OC4J data source contains connection information for map requests.
- To set up an OC4J data source, click the Admin key icon.



## Opening the MapViewer Home Page

Before MapViewer can start accepting map requests, a data source must be established with OC4J. MapViewer includes a Web page to help administer OC4J data sources.

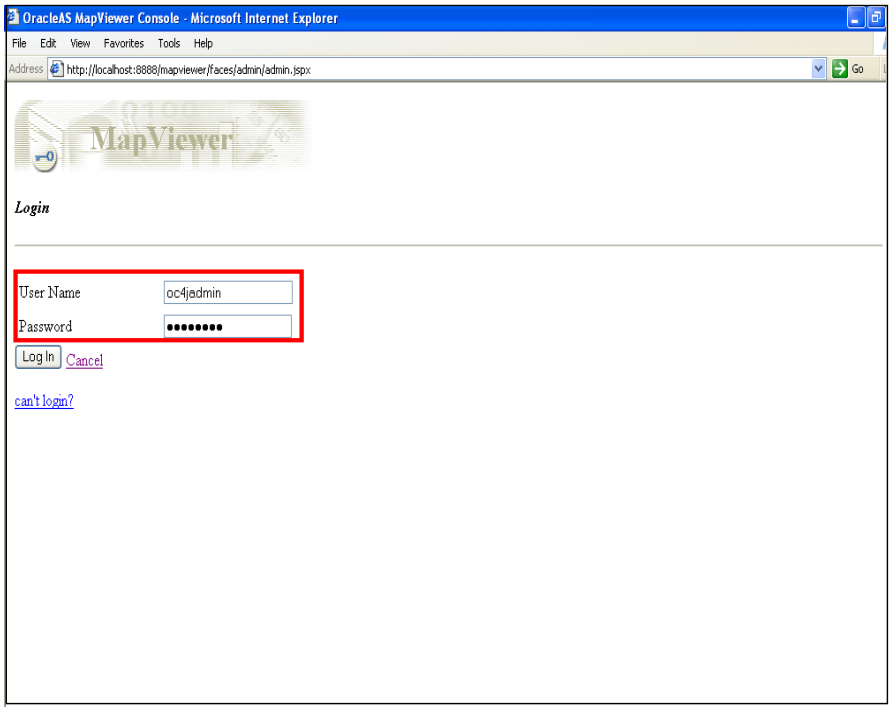
First, go to the MapViewer home page by specifying the following URL.

http://*localhost*:8888/mapviewer

If MapViewer is running on a different machine, replace *localhost* with the name of the server where MapViewer is running.

Then, click the Admin icon at the upper-right corner of the MapViewer home page.

# OC4J: Connecting as `oc4jadmin`




- Only an OC4J administrator can add a data source.
- You are prompted for a username and password.
- For a stand-alone OC4J installation, the username is `oc4jadmin`.
- The password was set up when OC4J was installed.

ORACLE

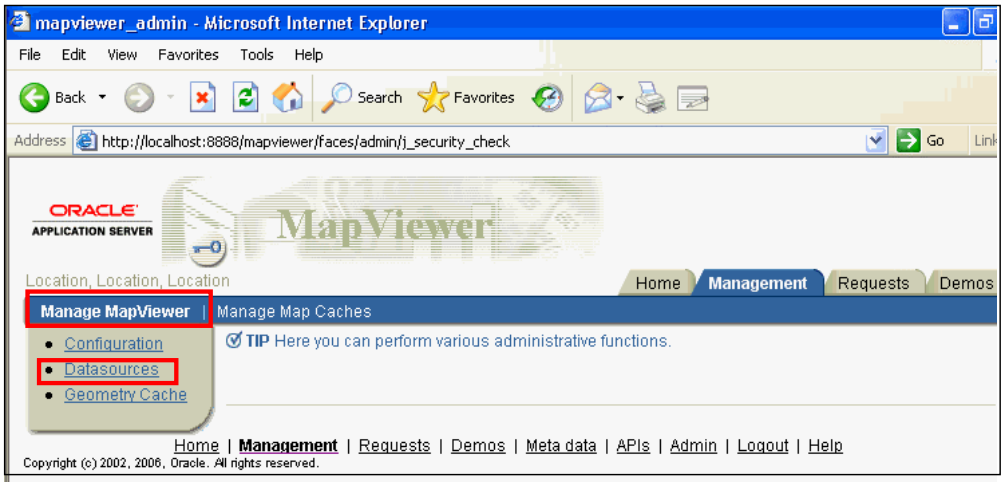


## OC4J: Connecting as `oc4jadmin`

You must be an authorized OC4J administrator to add a data source.
If MapViewer is installed through a stand-alone OC4J instance, the username is `oc4jadmin`.
The `admin` password was established when OC4J was initialized.

# OC4J: Adding a Data Source

1. Under the Management tab, click Manage MapViewer.
2. Click Datasources.




ORACLE

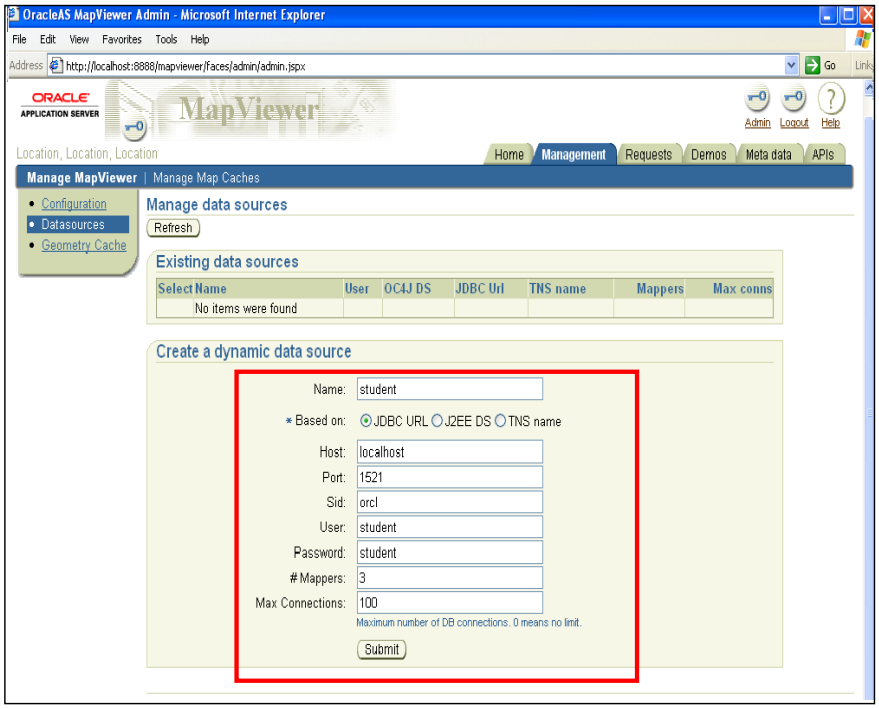


## OC4J: Adding a Data Source

On the MapViewer Admin page, perform the following steps:

1. Click the Management tab.
2. Click Manage MapViewer.
3. Click Datasources.

# OC4J: Adding a Data Source




- The data source contains connection information for map requests.
- Name: When prototyping, you can use the same name as `jdbc_user`.
- #Mappers: This specifies the number of map-rendering processes.
- After you click Submit, the newly added data source should appear under "Existing data sources."

ORACLE



## OC4J: Adding a Data Source (continued)

A data source contains connection information, which is required when you submit a map request to MapViewer.

When prototyping, you can name the data source using the same name as the Oracle user that you connect to. But in a production application, you may not want to expose the Oracle username that owns your map data.

When you add a data source, you can specify the number of "mappers." Mappers render processes that process one map request at a time. MapViewer manages the map request queue, and sends requests to available mappers.

After you click Submit, if the data source addition is successful, the data source appears under "Existing data sources" on the MapViewer Admin page.

# OC4J: Another Way to Add a Data Source

- The `MapViewerConfig.xml` file is located in:
- `%OC4J_HOME%\applications\mapviewer\web\WEB-INF\conf\MapViewerConfig.xml`.
- You can register data sources to automatically start when OC4J is started.
- Uncomment the example at the bottom of `MapViewerConfig.xml`:

```
<map_data_source name= "data_source_name"
                 jdbc_host="your_host_name"
                 jdbc_sid="orcl"
                 jdbc_port="1521"
                 jdbc_user="student"
                 jdbc_password="!student"
                 jdbc_mode="thin"
                 number_of_mappers= "3"
                 allow_jdbc_theme_based_foi="false"
/>
```

Note: The password (with leading !) is encrypted after the first time the data source is added.

ORACLE



## OC4J: Another Way to Add a Data Source

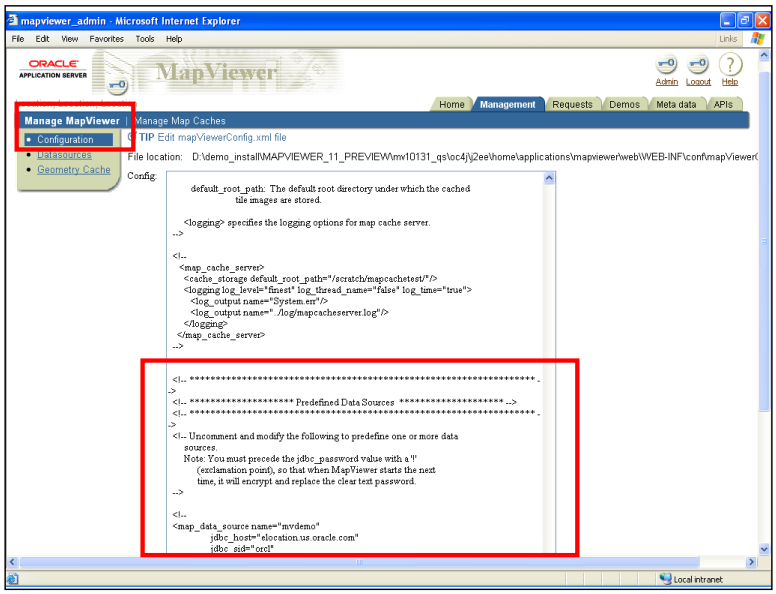
The `MapViewerConfig.xml` file contains initialization parameters for MapViewer. You can configure MapViewer to automatically add data sources at startup by including one or more `map_data_source` elements in the `MapViewerConfig.xml` file.

The password must include a leading exclamation point. After the first time the data source is added, the password is automatically encrypted in the `MapViewerConfig.xml` file.

The configuration file is located in `%OC4J_HOME%\applications\mapviewer\web\WEB-INF\conf\MapViewerConfig.xml`.

# Editing the MapViewer Config File on the MapViewer Admin Web Page

- `MapViewerConfig.xml` can also be edited on the MapViewer Admin Web page.
- MapViewer can also be restarted from the Admin Web page.




ORACLE



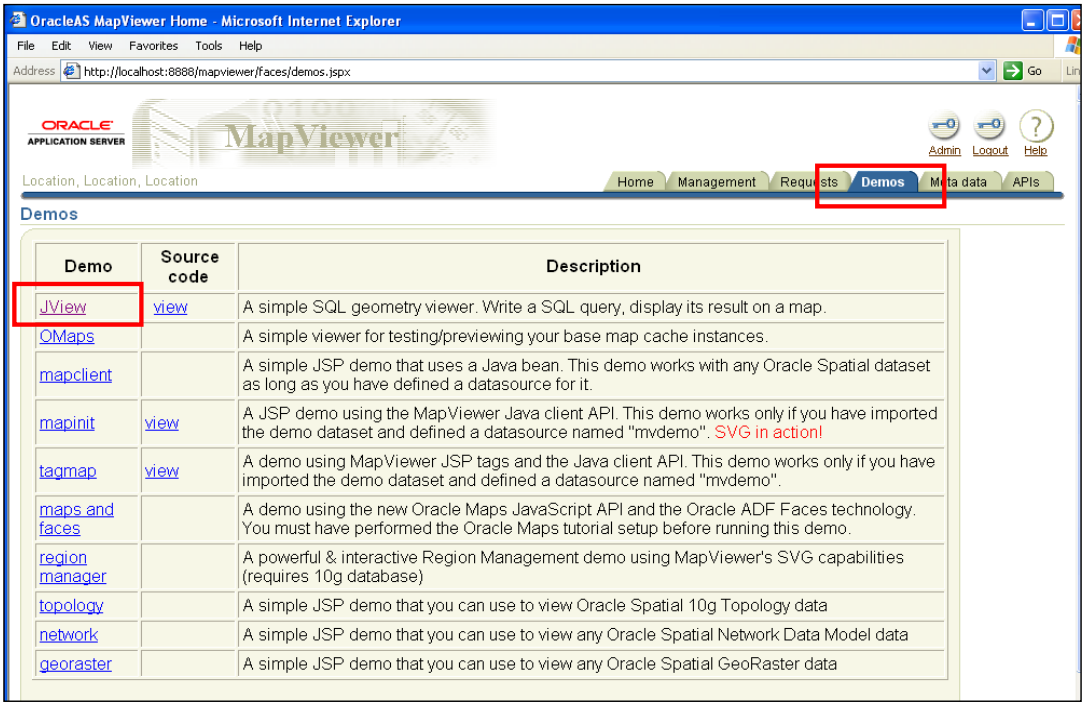
## Editing the MapViewer Config File on the MapViewer Admin Web Page

From the MapViewer Admin page, you can also edit the MapViewer configuration file and restart MapViewer so it can consider your edits. This enables you to modify MapViewer initialization parameters—for example, add a new data source—through the MapViewer Admin Web page.

# Lesson Agenda

- Introduction to MapViewer
- Architecture of Oracle Application Server MapViewer
- Installation of Oracle Application Server MapViewer
  - Adding an OC4J data source
- MapViewer demos
- Debugging

# MapViewer Welcome Page: Demos (`JView.jsp`: A Very Useful Demo)




ORACLE

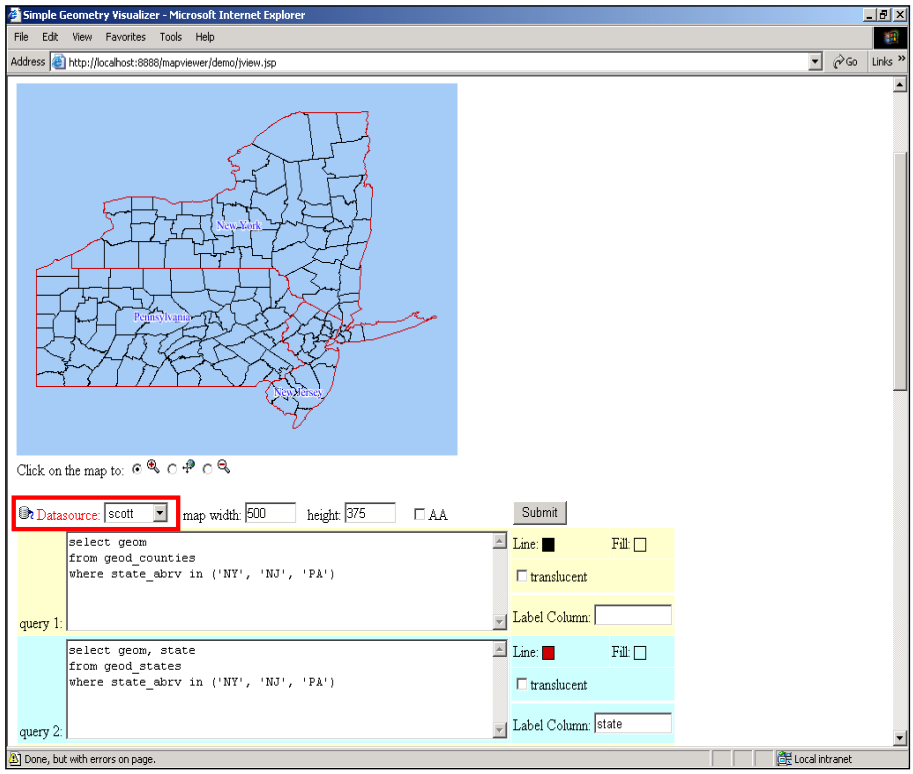


## MapViewer Welcome Page: Demos (`JView.jsp`: A Very Useful Demo)

The MapViewer Welcome page includes a Demos tab that lists several very useful JSP demonstrations.

`JView.jsp` is one of the most useful demonstrations. It enables you to submit up to three SQL statements that return `SDO_GEOMETRY` objects, and render the result of each.

# Running the `JView.jsp` Demo




- Executes up to three user-specified queries (do not include semicolon)
- Renders geometries returned from each query
- Requires you to select the data source from a drop-down list
- Allows you to control the color and label for each query result
- Allows you to zoom in, zoom out, and pan

ORACLE



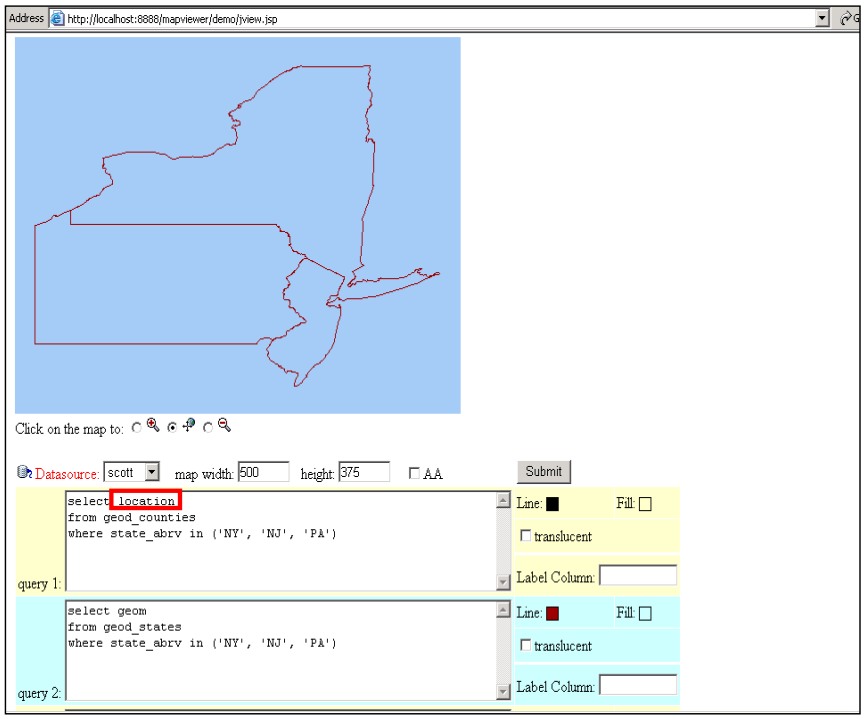
## Running the `JView.jsp` Demo

This is the `JView.jsp` page. As mentioned previously, there are places to submit up to three SQL statements that return `SDO_GEOMETRY` objects, and render the result of each.

Do not forget to pick the appropriate data source from the Datasource drop-down list.

Also, do not end the SQL statements with a semicolon.

# JView.jsp Demo: Error




- Counties did not render.
- A student incorrectly specifies `location` as the column name instead of `geom`.
- By default, errors are displayed in the console that OC4J was started in.

ORACLE



## JView.jsp Demo: Error

This example shows an attempt to render some of the counties and states in the U.S.

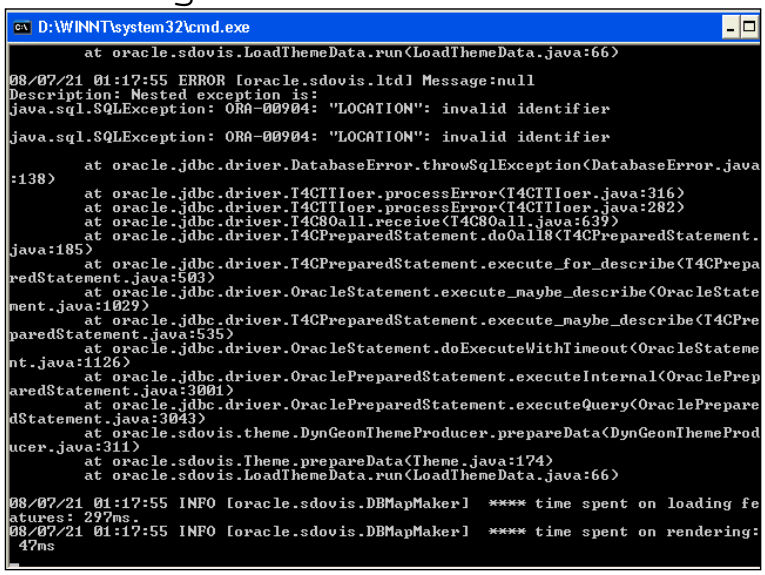
A student has incorrectly chosen a column name for counties; therefore, the counties are not rendered. The next slide shows the error message displayed in the OC4J console.

# Lesson Agenda

- Introduction to MapViewer
- Architecture of Oracle Application Server MapViewer
- Installation of Oracle Application Server MapViewer
- Adding an OC4J data source
- MapViewer demos
- Debugging

# Errors Reported in the OC4J Console (by Default)

- Scroll through the console to find where the error begins.
- `log_level` of reported errors can be configured in the `MapViewerConfig.xml` file.




ORACLE

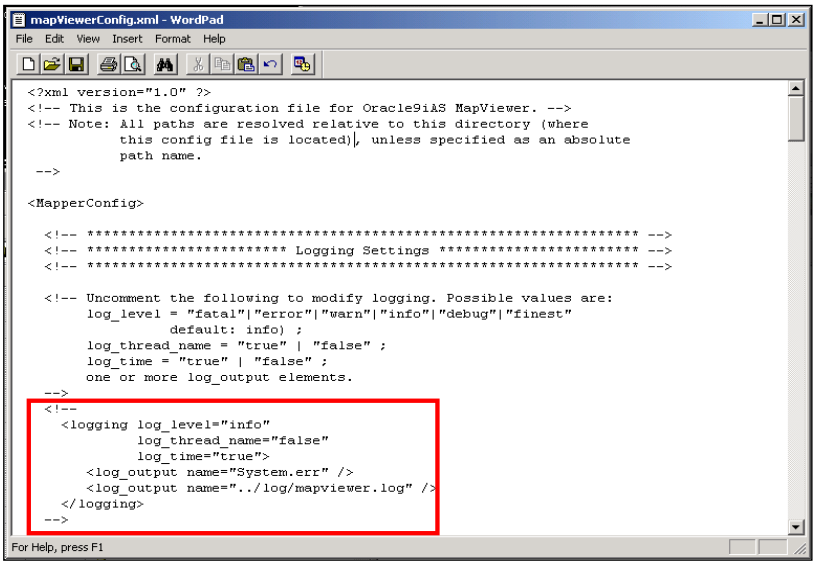


## Errors Reported in the OC4J Console (by Default)

By default, MapViewer reports errors in the console where OC4J was started. The error-logging level and location of the error log can be configured in the `MapViewerConfig.xml` file (see the following slide).

# MapViewerConfig.xml: Setting log_level

- The `logging` element can be uncommented.
- You can specify `log_level` and the `log_output` name.




ORACLE



## MapViewerConfig.xml: Setting log_level

`MapViewerConfig.xml` includes a commented section for logging errors. This can be uncommented, and you can set `log_level` and location for the log output. In this example, note that two `log_output` names are specified: one is `System.err`, which is the OC4J console, and the other is a file.

# Summary

In this lesson, you should have learned how to:

- Describe MapViewer
- Describe the components of MapViewer
- Install and configure OC4J with the MapViewer Quickstart kit
- Render query results with the `JView.jsp` demo
- Edit the `MapViewerConfig.xml` file to configure a data source
- Configure the logging level reported by MapViewer

ORACLE



## Summary

This lesson introduced the concepts associated with Oracle Application Server MapViewer. It described MapViewer and the components required to publish maps to the Web. Also, a sample JSP was used to retrieve a map on the Web.

# Practice 7: Overview

This practice covers the following topics:

- Installing OC4J and MapViewer
- Viewing data in MapViewer

ORACLE



## Practice 7: Overview

In this practice, you install OC4J and MapViewer. You also view the data in MapViewer.

## Practice 7

1. Extract the `mv10131_qs.zip` file under the root directory on the D: drive.
   **Note:** The `mv10131_qs.zip` file is located in the `D:\labs\software` folder. You can extract the `mv10131_qs.zip` file anywhere outside your `ORACLE_HOME`.
2. Initialize OC4J and deploy MapViewer. Edit the `start.bat` file in your `D:\mv10131_qs` folder. Deploy MapViewer by running `start.bat`. Choose `welcome` as the `oc4jadmin` password when prompted.
3. Create a dynamic data source. A data source defines the connection information to the database for map requests. Define a data source with the Oracle Application Server MapViewer Admin JSP.
4. View the data that you loaded in the lesson titled "Loading Spatial Data" by using MapViewer. Select the geometries from the `geod_states` and `geod_counties` tables.
5. Configure the `student` data source. Through the MapViewer Admin page, configure a data source that automatically starts when MapViewer starts.

# 8

# Indexing Spatial Data

ORACLE

# Objectives

After completing this lesson, you should be able to do the following:

- Explain the concept of spatial indexing
- Generate an R-tree spatial index
- Estimate the size of an R-tree index

ORACLE



## Objectives

In this lesson, you learn about spatial indexes. You learn how to define a spatial index on a spatial layer.

# Lesson Agenda

- Concepts of R-tree indexing
- `CREATE INDEX` and the R-tree parameters
- Analyze, drop, and alter operations on the spatial index
- Spatial index dictionary views
- Estimation of the R-tree index size and the resources required

ORACLE

# Indexing Spatial Data

- Spatial index is used to optimize spatial query performance.
- Oracle Spatial uses R-tree indexing for efficient access to data.
- Spatial indexes can be built on two, three, or four dimensions of data.
  - Default is two.
- Spatial index provides an exclusive and exhaustive coverage of spatial objects.
- All elements within a geometry including points, lines, and polygons are indexed.

ORACLE



## Indexing Spatial Data

Oracle Spatial uses a spatial index to optimize spatial query performance. In the same way that an index is created on scalar data to improve query performance, a spatial index improves spatial query performance.

Oracle Spatial uses R-tree indexes for efficient access to data. Spatial indexes can be built on two, three, or four dimensions of data. A spatial index can help you find:

- Objects within an indexed data space that interact with a given point or area of interest (query window)
- Pairs of objects from within two indexed data spaces that interact spatially with each other (spatial join)

# R-Tree Indexing

- R-tree indexing is based on minimum bounding rectangles (MBRs) for two-dimensional (2D) data or minimum bounding volumes (MBVs) for three-dimensional (3D) data.
- R-tree indexing is used to index spatial data.
  - Requires almost no tuning
- Each index entry approximates geometry by using MBR for 2D and MBV for 3D.
- MBRs and MBVs are indexed internally using a tree structure.

ORACLE



## R-Tree Indexing

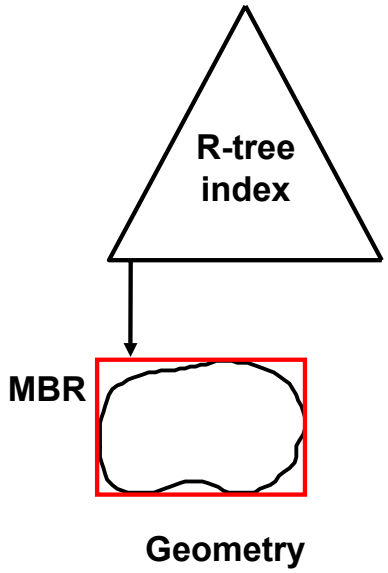
R-tree indexing is the way spatial data is indexed in Oracle Spatial. R-tree indexes are simple to create, and there is very little or no tuning required to achieve optimal performance. R-tree indexes can be created on two, three, or four dimensions of spatial data.

R-tree indexes are based on minimum bounding rectangles for two-dimensional (2D) data. This means that a rectangle is built that approximates each geometry based on the minimum and maximum values for each dimension. When indexing three-dimensional (3D) data, each index entry is based on a minimum bounding volume around each geometry.

After all the minimum bounding rectangles (or volumes) are built, the approximations are sorted and an index tree structure is built and written to a table. The table structure is the index.

# R-Tree Indexing Concept

Each leaf node of the R-tree index stores an MBR and a pointer to the original geometry.

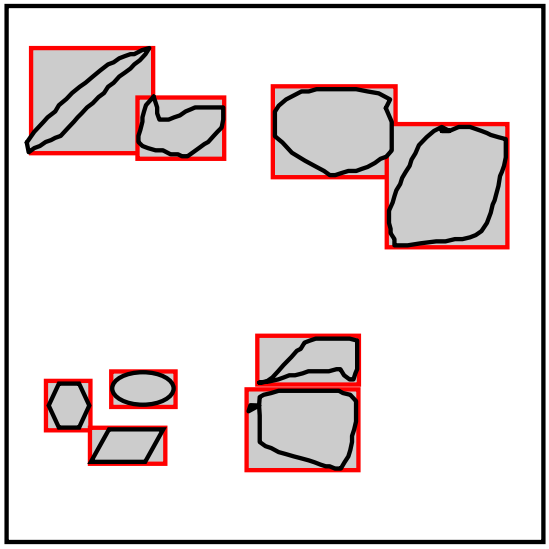





## R-Tree Indexing Concept

Starting with a geometry, the R-tree builds a minimum bounding rectangle around that geometry.

# How Geometries Are Indexed with R-Trees

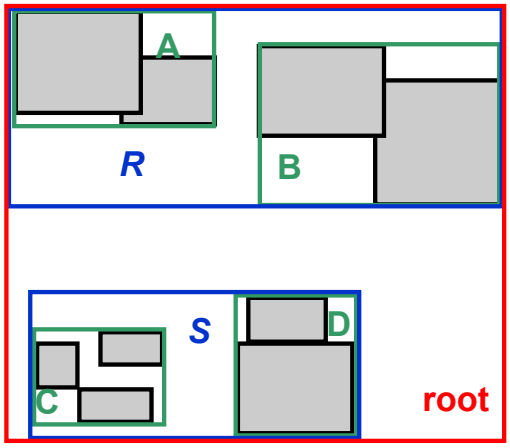


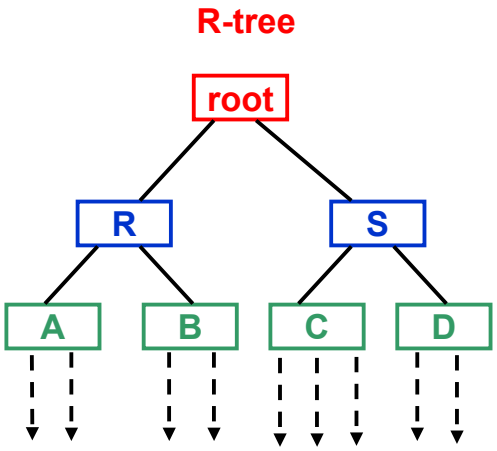


## How Geometries Are Indexed with R-Trees

When an R-tree index is built, an MBR is built around each geometry. In this example, there are nine minimum bounding rectangles created, one for each geometry. At the lowest level of an R-tree index, there is a geometry pointer (to the `ROWID` of the geometry) and the MBR associated with that geometry. R-tree indexes have one index entry (or leaf node) for each geometry.

## How Geometries Are Indexed with R-Trees




MBRs, pointers to geometries

- Geometries close to each other are grouped together in an MBR and stored on a higher level in the R-tree.
- Each node in the R-tree has the same number of branches.

ORACLE



### How Geometries Are Indexed with R-Trees (continued)

After the MBRs are built around each geometry, the approximations are grouped such that geometries that are close to each other end up close to each other in the R-tree index structure. For example, the geometries close to each other are grouped together in A. Similar groupings are shown for B, C, and D. At this lowest level (or *leaf* level) of the R-tree, A, B, C, and D contain MBRs for each of the geometries grouped together as well as pointers to each of those geometries.

Next, A, B, C, and D are grouped together based on their proximity. In this case, A and B are grouped together into *R*, and C and D are grouped together into *S*. The contents of *R* include the minimum bounding rectangles around A and B, as well as pointers to A and B.

The top level (or *root*) of the R-tree index contains pointers to *R* and *S* as well as the minimum bounding rectangles associated with both of those nodes.

**Note:** The R-tree structure is optimized for accessing elements that are close to each other.

The number of branches for an R-tree index is automatically calculated based on the database block size used for the instance, as well as the number of dimensions to be indexed. Each node in the R-tree has the same number of branches. The root node contains pointers to between 25 and 50 minimum bounding rectangles, and each of the 25–50 minimum bounding rectangles would have pointers to 25–50 other minimum bounding rectangles, and so on until the leaf nodes are reached, each of which would have pointers to 25–50 geometries and their MBRs.

# Lesson Agenda

- Concepts of R-tree indexing
- **`CREATE INDEX`** and the R-tree parameters
- Analyze, drop, and alter operations on the spatial index
- Spatial index dictionary views
- Estimation of the R-tree index size and the resources required

## CREATE INDEX Syntax

```
CREATE INDEX <index-name>
  ON <table-name> (<column-name>)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  [PARAMETERS (
   'SDO_RTR_PCTFREE = <param_value>
    <storage_parameters> = <param_value> ... ')
  ] [PARALLEL [<parallel_degree>]];
```

- <INDEX-NAME>: Name of index
- <TABLE-NAME>: Table that contains a column of the SDO_GEOMETRY type
- <COLUMN-NAME>: Name of the column of the SDO_GEOMETRY type

  Add the INDEXTYPE IS MDSYS.SPATIAL_INDEX clause.

ORACLE



### CREATE INDEX Syntax

The CREATE INDEX statement in the slide is exactly the same as any Oracle CREATE INDEX statement, with some additional command options added.

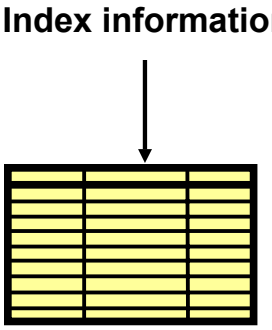
The only clause to add is INDEXTYPE IS MDSYS.SPATIAL_INDEX. Optionally, other parameters can be specified as well.

The following list details the command options:

- <INDEX-NAME>: Name of the index. Normal Oracle index-naming conventions apply to the index name. The length of the name must be 30 characters or fewer, and the index name must start with an alphabetic character.
- <TABLE-NAME>: Name of the table that contains the spatial column to be indexed. The spatial column is of the SDO_GEOMETRY type.
- <COLUMN-NAME>: The name of the column of the SDO_GEOMETRY type that is to be spatially indexed

# Creating a Spatial Index: Example

```
CREATE INDEX geod_states_sidx
   ON geod_states (geom)
 INDEXTYPE IS mdsys.spatial_index;
```






## Creating a Spatial Index: Example

When an R-tree spatial index is built, an Oracle table is created. This is also known as the spatial index table. The spatial index table is visible to the end user, and is flagged in Oracle's metadata tables (`USER_TABLES`, `ALL_TABLES`, `DBA_TABLES`) as `SECONDARY = 'Y'`.

The tablespace for the spatial index appears as `NULL` if it is looked up in `USER_INDEXES`. This is because the spatial index name is only a logical structure associated with another physical structure (the spatial index table). The tablespace and storage parameters can be specified for the spatial index table associated with the spatial index name.

**Note:** Do not update, alter, or delete the spatial index table. When you drop the index, this table is automatically dropped.

## CREATE INDEX: R-Tree Parameters

- <SDO_INDX_DIMS>:
  - Defines the number of dimensions on which the index is built
  - Enables indexing of more than two dimensions
- <SDO_RTR_PCTFREE>:
  - Is the percentage of R-tree branches reserved in each index node to accommodate future insertions
  - Accepts 10 percent as the default value—that is, 10 percent of the branches in the tree are reserved when the spatial index is created

ORACLE



### CREATE INDEX: R-Tree Parameters

- <SDO_INDX_DIMS>: Number of dimensions to spatially index. The default is two dimensions.
- <SDO_RTR_PCTFREE>: Percentage of branches to reserve in the R-tree to accommodate future insertions or updates. The efficiency of an R-tree index may degrade over time if a lot of data is inserted or updated. Specifying SDO_RTR_PCTFREE is a way to reduce the effects of inserting or updating spatial data that has an R-tree index. For instance, if it is anticipated that the spatial layer will get 50 percent more data over time, specifying SDO_RTR_PCTFREE=50 leaves room in the index for that data. The default value for SDO_RTR_PCTFREE is 10 percent—that is, 10 percent of the slots are left empty to accommodate future inserts and updates.

## CREATE INDEX: R-Tree Parameters

- <LAYER_GTYPE>:
  - Is the layer type constraint for all spatial index types
  - Allows only geometries of the given type to be indexed
  - Is used for performance and geometry type checking
- Valid values are:
  - POINT
  - LINE or CURVE
  - POLYGON
  - MULTIPOINT
  - MULTILINE or MULTICURVE
  - MULTIPOLYGON
  - COLLECTION (DEFAULT, same as no type constraint)
- Multitypes encapsulate single types.

ORACLE



### CREATE INDEX: R-Tree Parameters (continued)

- **<LAYER_GTYPE>:** A geometry type constraint applied to each geometry as it is indexed. It specifies that only geometries of the given type are allowed in a spatial layer with an index.
- There are two benefits of having geometry type constraints: first, to prevent data from being inserted into a spatial layer if it is not of a predefined type; and second, to make use of the type of information for internal query optimizations. The valid values for LAYER_GTYPE are specified as POINT, LINE, CURVE, POLYGON, MULTIPOINT, MULTILINE, MULTICURVE, MULTIPOLYGON, or COLLECTION. LINE and CURVE are synonyms, as are MULTILINE and MULTICURVE.
- Single types are allowed in their multitype counterparts (for example, single POINT is valid in MULTIPOINT LAYER_GTYPE).
  **Note:** The reverse is not true. You cannot store a MULTIPOINT geometry in a spatial layer, where the spatial index was built specifying LAYER_GTYPE=POINT.

## CREATE INDEX: R-Tree Parameters

<STORAGE_PARAMETERS>:

- INITIAL: Size of the initial extent of the spatial index table
- TABLESPACE: Name of the tablespace that contains the spatial index table
- WORK_TABLESPACE: Name of the tablespace where work tables are created during the creation of the R-tree index

ORACLE



### CREATE INDEX: R-Tree Parameters (continued)

- <STORAGE PARAMETERS>: Can be specified in the parameters string to provide information to the Oracle database about storage of the spatial index table
- The following can be specified:
  - INITIAL: Size of the initial extent of the spatial index table
  - TABLESPACE: Name of the tablespace that contains the spatial index table
  - WORK_TABLESPACE: Name of the tablespace where work tables are created during R-tree index creation. When an R-tree index is created, work tables are created and dropped that may consume a significant amount of space. Users or applications can control where that extra space is written to via the <WORK_TABLESPACE> spatial parameter of the CREATE INDEX command. This tablespace can be dropped later to reclaim space associated with the CREATE INDEX command. By putting these work tables in a separate tablespace, the separate tablespace can be managed independently.
- Some more storage parameters (not used often):
  - NEXT: Size of the next extent of the spatial index table
  - PCTINCREASE: Percentage by which to increase the previously used extent, when the need to create a new extent arises
  - MINEXTENTS: Minimum number of extents to create for the spatial index table
  - MAXEXTENTS: Maximum number of extents to create for the spatial index table

## CREATE INDEX: R-Tree Parameters

<SDO_DML_BATCH_SIZE>:

- Is a tuning parameter to batch index updates
- Enhances performance by making a large number of index changes in a single underlying index update during COMMIT processing
- Accepts 1000 as the default value, which is fine for most workloads
- Can be set higher if there is plenty of memory, and the workload performs more than 1000 INSERT, UPDATE, or DELETE operations between commits
  - 5000 is the maximum recommended value. Very little performance improvement has been observed beyond this value.

ORACLE



### CREATE INDEX: R-Tree Parameters (continued)

<SDO_DML_BATCH_SIZE>: Tuning parameter for batching index updates. It enhances performance when users perform a large number of INSERT, UPDATE, or DELETE operations between commits by allowing a large number of index updates to occur in the context of a single index update.

The default is 1000, which batches up to 1,000 index updates at a time during the COMMIT operation. If the workload requires more than 1,000 INSERT, UPDATE, or DELETE operations between commits, you can set <SDO_DML_BATCH_SIZE> higher.

**Note:** It is not recommended that you use quadtree indexes. You must always use R-tree indexes. Parameters such as SDO_LEVEL, SDO_COMMIT_INTERVAL, SDO_NUMTILES, and SDO_MAXLEVEL are quadtree index parameters. If you see any of these parameters being used, remove the listed parameters and create an R-tree index.

## CREATE INDEX: R-Tree Parameters

- <SDO_NON_LEAF_TBL>: This allows the R-tree index to be broken into two during index creation.
  - Nonleaf table (smaller table)
  - Leaf table (ROWID and MBR)
- When set to TRUE, the DBA can pin an R-tree index table in memory.
- A nonleaf table can be pinned in memory.
- Leaf table blocks are cached as they are accessed via ordinary Oracle data block caching.
- To pin the nonleaf index table, use:

```
ALTER TABLE <table_name> STORAGE(BUFFER_POOL
  KEEP);
```

ORACLE



### CREATE INDEX: R-Tree Parameters (continued)

In a limited set of workloads and applications, a DBA may pin a table in memory for optimal performance. An R-tree index table is like any other table, and can be pinned in memory.

R-tree index tables can be very large, making it impractical to pin the entire R-tree index table in memory.

Specifying SDO_NON_LEAF_TBL=TRUE in the parameter list of the CREATE INDEX statement causes the index table to be split into two pieces:

- A *nonleaf index table*, which is the smaller of the two tables and contains all the higher-level R-tree nodes (except for the leaf nodes). Because this table is always accessed during spatial queries, caching it in memory can help query performance.
- A *leaf index table*, which contains the ROWID and MBR for each of the geometries indexed

The DBA can then pin the nonleaf table in memory, allowing the leaf table to be cached with ordinary Oracle data block caching.

To pin the nonleaf index table, use the following statement:

```
ALTER TABLE <table_name> STORAGE(BUFFER_POOL KEEP);
```

The table is pinned in memory when the spatial data is accessed.

# R-Tree Nonleaf Index Table: Example

```
-- Create the index
CREATE INDEX geod_counties_sidx
  ON geod_counties(geom)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  PARAMETERS ('sdo_non_leaf_tbl=TRUE');

-- Find the non leaf index table name
SELECT sdo_nl_index_table
FROM user_sdo_index_metadata
WHERE sdo_index_name='GEOD_COUNTIES_SIDX';
----------
MDNT_A930$

-- Pin the table in memory
ALTER TABLE MDNT_A930$ STORAGE(BUFFER_POOL KEEP);
```

ORACLE



## R-Tree Nonleaf Index Table: Example

This slide shows an example of a `CREATE INDEX` statement with a spatial parameter `<SDO_NON_LEAF_TBL>` that forces the creation of a nonleaf index table. The example also demonstrates how to pin the nonleaf index table in memory.

## CREATE INDEX: PARALLEL

PARALLEL [<parallel_degree>]

- This breaks the work of index creation into smaller pieces that can be performed in parallel.
- <parallel_degree> is an optional parameter that specifies the degree of parallelism.
- If the degree of parallelism is not specified, Oracle RDBMS chooses a default on the basis of internal Oracle algorithms based on the number of CPUs.

```
CREATE INDEX geod_counties_sidx
 ON geod_counties(geom)
   INDEXTYPE IS MDSYS.SPATIAL_INDEX
   PARALLEL 4;
```

ORACLE

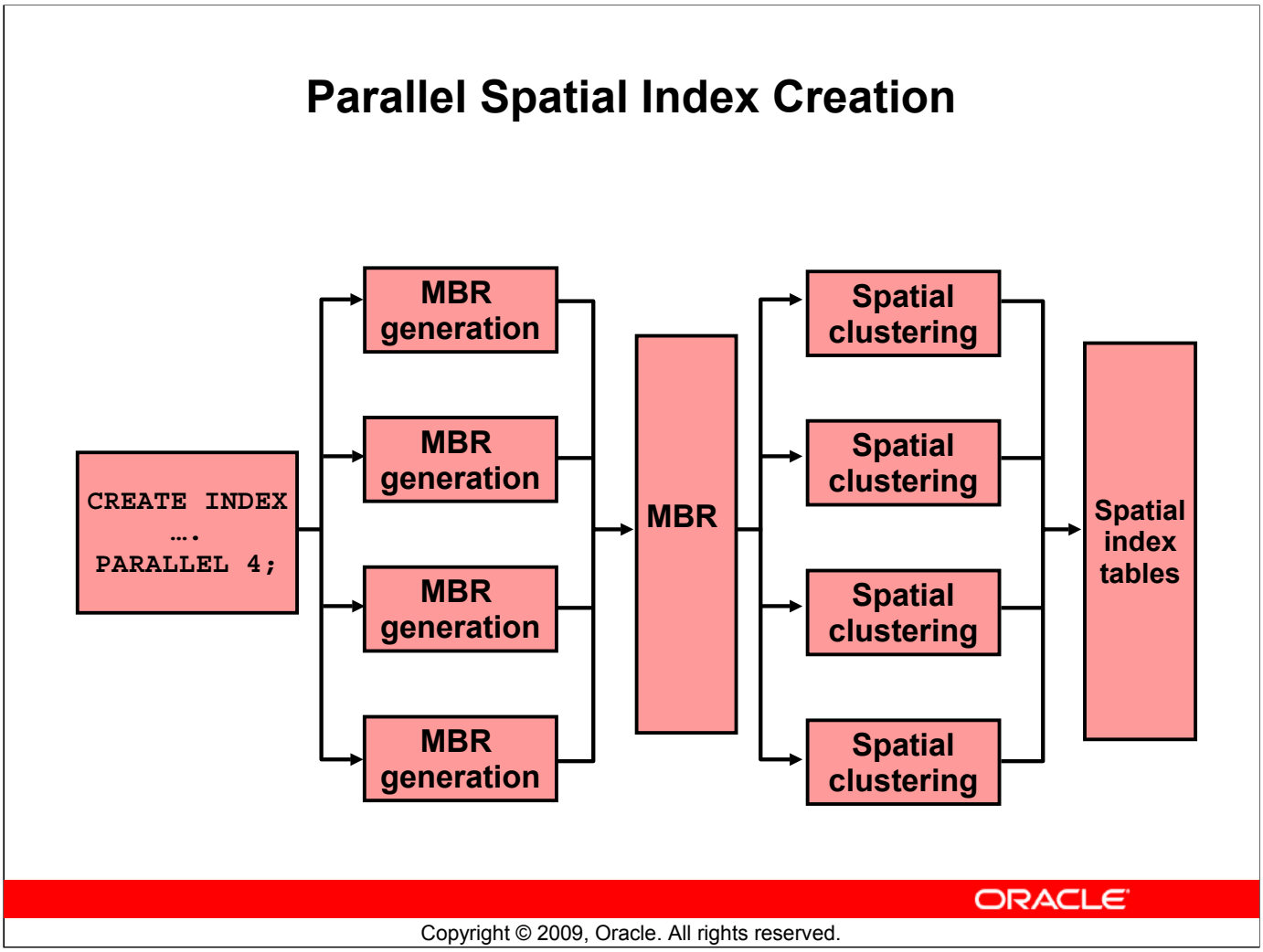


### CREATE INDEX: PARALLEL

The PARALLEL indexing parameter creates the spatial index with parallel processing.

If the PARALLEL keyword is not followed by an integer, the Oracle RDBMS determines the degree of parallelism (the number of simultaneous processes to use to create the index) by looking at the number of CPUs available to the Oracle instance, and multiplying that value by the PARALLEL_THREADS_PER_CPU initialization parameter.

If the PARALLEL keyword is followed by an integer, the Oracle RDBMS uses that value as the degree of parallelism.



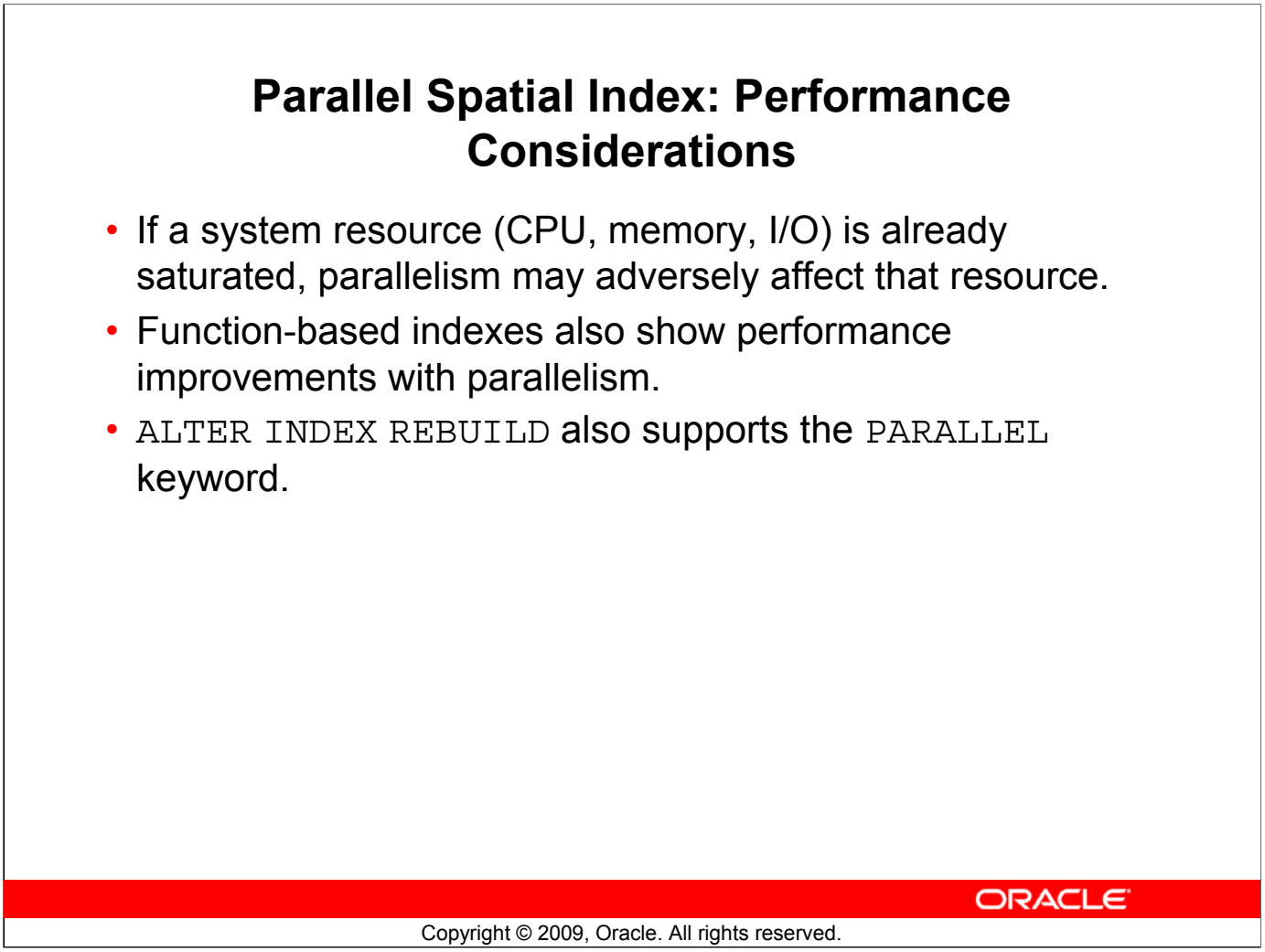

## Parallel Spatial Index Creation

Parallel index creation is enabled by specifying the `PARALLEL` keyword in the `CREATE INDEX` statement.

The following describes the process to create a parallel index:

- When parallel R-tree spatial index creation begins, multiple processes are started that generate MBRs for the spatial data in parallel. Multiple processes write the MBR information to a table, which is used for R-tree clustering operations.
- The `CREATE INDEX` command gathers all the MBR information to the temporary storage.
- Next, where possible, clustering operations are done in parallel. Clustering groups MBRs close in proximity, close to each other in the R-tree index structure.
- The result of R-tree clustering operations is the R-tree spatial index table.

Parallel index creation with R-trees is supported for 2D, 3D, and 4D data; geodetic indexes; and function-based indexes.




## Parallel Spatial Index: Performance Considerations

The following are some performance considerations regarding parallel spatial indexes:

- If a system has resources that are saturated, creating parallel spatial indexes may worsen those saturation issues.
  - If a system is paging and/or swapping because memory is highly used, creating multiple threads to create a spatial index will use more memory and will cause more paging or swapping.
  - If a system's CPU utilization is already very high, creating indexes in parallel may cause less work to be done due to context switching between processes.
- Data sets that have large, complicated polygons benefit more from parallel index creation than point-only data sets.
- Spatial function-based index creation can also benefit from parallelism.

The `ALTER INDEX REBUILD` syntax also supports the `PARALLEL` keyword.

# Lesson Agenda

- Concepts of R-tree indexing
- `CREATE INDEX` and the R-tree parameters
- Analyze, drop, and alter operations on the spatial index
- Spatial index dictionary views
- Estimation of the R-tree index size and the resources required

ORACLE

# Analyzing Spatial Tables

```
exec dbms_stats.gather_table_stats
      (<schema_name>,<table_name>)
```

- Manually gather statistics on the table that the index was built on with the DBMS_STATS package.
- Information is gathered for the Oracle optimizer:
  - Information gathered includes the number of rows and indexes, and other information.
  - Optimizer creates optimal query execution plans by using the gathered statistics.
- There is no need to gather statistics on spatial index tables.

ORACLE



## Analyzing Spatial Tables

After creating and loading a table with a spatial layer, and after creating all indexes including the spatial index, you are recommended to gather statistics on the table. Information about the table, including the number of rows and indexes that exist on the table, are gathered. Oracle Optimizer uses this information to generate an efficient execution plan for a SQL statement.

Gathering table statistics is not specific to Oracle Spatial. Oracle recommends gathering statistics for all user-defined tables with the DBMS_STATS PL*SQL package. Analyzing statistics on tables is performed automatically, so you do not need to perform this manually. However, if you want to do it manually, you can use the syntax as shown in the slide.

**Note:** It is not necessary to gather statistics on spatial index tables.

## DROP INDEX Syntax

- This drops the spatial index and the associated spatial index table.
- If CREATE INDEX on an SDO_GEOMETRY column does not successfully complete, try DROP INDEX. If that does not work, call DROP INDEX by using the FORCE option.

```
DROP INDEX <index_name>;
```

```
DROP INDEX <index_name> FORCE;
```



### DROP INDEX Syntax

Spatial indexes created with the CREATE INDEX command can be dropped with the DROP INDEX command. Dropping a spatial index drops the spatial index table. You must never manually drop spatial index tables.

Sometimes CREATE INDEX on a column of the SDO_GEOMETRY type does not successfully complete. If DROP INDEX index_name does not work, the FORCE option must be used. For example:

```
DROP INDEX index_name FORCE;
```

## ALTER INDEX REBUILD: Syntax

```
ALTER INDEX <index_name>
REBUILD
[PARAMETERS (<parameter string>)];
```

- This rebuilds a spatial index table.
- Parameters are passed in as a quoted string.
  - For example, to change the spatial index table tablespace:
    - 'Tablespace = indx_tblspc'
    - 'Work_tablespace = work_indx_tblspc'

ORACLE



### ALTER INDEX REBUILD: Syntax

The ALTER INDEX syntax can be used with Oracle Spatial indexes. ALTER INDEX with the REBUILD option allows a spatial index table to be rebuilt, possibly with different parameters than were specified when the spatial index table was built. The REBUILD command option causes the entire spatial index to be rebuilt. All parameters associated with the index can be changed in the parameter list of ALTER INDEX REBUILD. For instance, the tablespace and work tablespace can be specified again during a rebuild. Also, LAYER_GTYPE can be specified again.

## ALTER INDEX REBUILD ONLINE: Syntax

```
ALTER INDEX <index_name>
REBUILD ONLINE
[PARAMETERS (<parameter string>)];
```

- This rebuilds the index without blocking the index.
- Only queries are permitted during an ONLINE rebuild.
  - INSERT, UPDATE, and DELETE operations that would affect the index are blocked while the index is rebuilding.
- You cannot use the ONLINE keyword for a rebuild if the index:
  - Was created using the 'SDO_NON_LEAF_TBL=TRUE' parameter
  - Is a partitioned spatial index

ORACLE



### ALTER INDEX REBUILD ONLINE: Syntax

- Only queries are permitted during an online rebuild.
- INSERT, UPDATE, and DELETE operations that affect the index are blocked while the index is rebuilding.

You cannot use the ONLINE keyword to rebuild the spatial index if:

- The index was created using the 'SDO_NON_LEAF_TBL=TRUE' parameter
- The index is a partitioned spatial index

The ALTER INDEX REBUILD statement does not use any previous parameters from the index creation. All parameters must be specified for the index that you want to rebuild.

## ALTER INDEX RENAME TO: Syntax

```
ALTER INDEX <index_name>
RENAME TO
<new_index_name>;
```

- Changes the name of the spatial index
- Can rename only the index and not spatial index tables

ORACLE



### ALTER INDEX RENAME TO: Syntax

`ALTER INDEX` with the `RENAME TO` clause renames the index. It renames only the spatial index and not the spatial index table.

# Lesson Agenda

- Concepts of R-tree indexing
- `CREATE INDEX` and the R-tree parameters
- Analyze, drop, and alter operations on the spatial index
- Spatial index dictionary views
- Estimation of the R-tree index size and the resources required

ORACLE

# Spatial Index Dictionary Views

- Two views that maintain metadata about spatial indexes:
  - `USER_SDO_INDEX_METADATA`: Shows all the spatial index metadata for the current user
  - `ALL_SDO_INDEX_METADATA`: Shows all the spatial index metadata that the current user has the `SELECT` privilege for
- Oracle Spatial automatically maintains spatial index dictionary views.

ORACLE





## Spatial Index Dictionary Views

Two views that contain the metadata associated with spatial indexes are defined. They are:

- `USER_SDO_INDEX_METADATA`
- `ALL_SDO_INDEX_METADATA`

These views contain all the index metadata associated with spatial indexes. Unlike the `*_SDO_GEOM_METADATA` views where the user is responsible for populating the views, the `*_SDO_INDEX_METADATA` views are entirely maintained by Oracle Spatial.

The `USER_SDO_INDEX_METADATA` view shows all the spatial index metadata for the current user. The `ALL_SDO_INDEX_METADATA` view shows all the spatial index metadata that the current user has the `SELECT` privilege for.

## USER_SDO_INDEX_METADATA View

| | |
|---|---|
| SDO_INDEX_OWNER | VARCHAR2(32) |
| SDO_INDEX_TYPE | VARCHAR2(32) |
| SDO_INDEX_NAME | VARCHAR2(32) |
| SDO_INDEX_TABLE | VARCHAR2(32) |
| SDO_INDEX_PRIMARY | NUMBER |
| SDO_INDEX_PARTITION | VARCHAR2(32) |
| SDO_PARTITIONED | VARCHAR2(32) |
| SDO_TSNAME | VARCHAR2(32) |
| SDO_COLUMN_NAME | VARCHAR2(2048) |
| SDO_INDEX_DIMS | NUMBER |
| SDO_RTREE_HEIGHT | NUMBER |
| SDO_RTREE_NUMNODES | NUMBER |
| SDO_RTREE_DIMENSIONALITY | NUMBER |
| SDO_RTREE_FANOUT | NUMBER |
| SDO_RTREE_ROOT | VARCHAR2(32) |
| SDO_RTREE_SEQ_NAME | VARCHAR2(32) |
| SDO_RTREE_PCTFREE | NUMBER |
| SDO_INDEX_STATUS | VARCHAR2(32) |
| SDO_LAYER_GTYPE | VARCHAR2(32) |

| | |
|---|---|
| SDO_LEVEL | NUMBER |
| SDO_NUMTILES | NUMBER |
| SDO_MAXLEVEL | NUMBER |
| SDO_COMMIT_INTERVAL | NUMBER |
| SDO_FIXED_META | RAW(255) |
| SDO_TABLESPACE | VARCHAR2(32) |
| SDO_INITIAL_EXTENT | VARCHAR2(32) |
| SDO_NEXT_EXTENT | VARCHAR2(32) |
| SDO_PCTINCREASE | NUMBER |
| SDO_MIN_EXTENTS | NUMBER |
| SDO_MAX_EXTENTS | NUMBER |
| SDO_RTREE_QUALITY | VARCHAR2(32) |
| SDO_INDEX_VERSION | NUMBER |
| SDO_INDEX_GEODETIC | VARCHAR2(8) |
| SDO_NL_INDEX_TABLE | VARCHAR2(32) |
| SDO_DML_BATCH_SIZE | NUMBER |
| SDO_RTREE_ENT_XPND | NUMBER |
| SDO_ROOT_MBR | SDO_GEOMETRY |

ORACLE



### USER_SDO_INDEX_METADATA View

The slide shows the columns associated with the USER_SDO_INDEX_METADATA view. Note that this view does not have information such as the table name and the column name on which the index was built.

# Spatial Index Informational Views

Useful spatial index metadata information can also be extracted from the following views:

- USER_SDO_INDEX_INFO
- ALL_SDO_INDEX_INFO

| | |
|---|---|
| INDEX_NAME | VARCHAR2(32) |
| TABLE_NAME | VARCHAR2(32) |
| COLUMN_NAME | VARCHAR2(2048) |
| SDO_INDEX_TYPE | VARCHAR2(32) |
| SDO_INDEX_TABLE | VARCHAR2(32) |

ORACLE



## Spatial Index Informational Views

Conveniently, USER_SDO_INDEX_INFO and ALL_SDO_INDEX_INFO views provide useful spatial index metadata information. Otherwise, to get the same information, you would need to join different Oracle dictionary views.

# Renaming a Table with Spatial Data

```
ALTER TABLE <old_table_name>
RENAME TO <new_table_name>;
```

- If the renamed table has one or more spatial indexes, the index metadata is automatically updated.
- You must manually update the `USER_SDO_GEOM_METADATA` entry.

```
UPDATE user_sdo_geom_metadata
SET table_name = 'NEW_TABLE_NAME'
WHERE table_name = 'OLD_TABLE_NAME';
```

ORACLE



## Renaming a Table with Spatial Data

You can rename tables with `SDO_GEOMETRY` columns in the same way as any Oracle table can be renamed. If the renamed table has one or more spatial indexes, the index metadata is automatically updated to reflect the changes. However, you are responsible for keeping the geometry metadata (entries in the `USER_SDO_GEOM_METADATA` view) up to date. The slide shows an example of keeping `USER_SDO_GEOM_METADATA` synchronized with a changed table name.

# Lesson Agenda

- Concepts of R-tree indexing
- `CREATE INDEX` and the R-tree parameters
- Analyze, drop, and alter operations on the spatial index
- Spatial index dictionary views
- Estimation of the R-tree index size and the resources required

ORACLE

## ESTIMATE_RTREE_INDEX_SIZE Function

- This estimates the maximum number of megabytes needed for an R-tree spatial index table.
- During index creation, the actual space requirement may be three times the returned value.

```
SIZE(Mb) = SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE
           (SCHEMA, TABLE_NAME,
            COLUMN_NAME [,PARTITION_NAME])
or
SIZE(Mb) = SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE
          (NUMBER_OF_GEOMS, DB_BLOCK_SIZE
          [,SDO_RTR_PCTFREE] [,NUM_DIMENSIONS]
          [,IS_GEODETIC])
```

ORACLE



### SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE Function

The SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE function returns the estimated maximum number of megabytes needed for the spatial index table for an R-tree spatial index to be created. The value returned is the maximum number of megabytes needed after index creation. During index creation, approximately three times this value of megabytes is needed in the tablespace to ensure that there is enough space for temporary tables while the index is created.

These are the parameters for SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE:

- SCHEMA_NAME: Owner of the table
- TABLE_NAME: Name of the table
- COLUMN_NAME: Name of the column of the SDO_GEOMETRY type in the TABLE_NAME table
- PARTITION_NAME: Name of the partition to estimate the index size for
- NUMBER_OF_GEOMS: Number of rows in the table
- DB_BLOCK_SIZE: DB_BLOCK_SIZE of the tablespace that contains the index table
- SDO_RTR_PCTFREE: Amount of free space to be left in the index table (default: 10)
- NUM_DIMENSIONS: Number of dimensions to be indexed (default: 2)
- IS_GEODETIC: 1 if the data being indexed is geodetic; otherwise, 0 (default: 0)

## ESTIMATE_RTREE_INDEX_SIZE Function: Examples

- Estimate the size of an R-tree index for user SCOTT on the GEOD_COUNTIES table with the GEOM column:

```
SELECT sdo_tune.estimate_rtree_index_size
         ('scott','geod_counties','geom')
FROM dual;
```

- Estimate the size of an R-tree index on a table that has 250,000 geometries, DB_BLOCK_SIZE set at 8 KB, SDO_RTR_PCTFREE set at 15, two-dimensional data, and a geodetic SRID:

```
SELECT sdo_tune.estimate_rtree_index_size
         (250000, 8192, 15, 2, 1)
FROM dual;
```

ORACLE



### ESTIMATE_RTREE_INDEX_SIZE Function: Examples

The examples in the slide show the usage of the SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE function for a given table that is already loaded. The second example shows how you can use the same function to estimate the size of an index table for which even the geometry table may not exist, but you have projected numbers for all the parameters.

## R-Tree Index Sizing: Usage Notes and WORK_TABLESPACE

- The number returned estimates the size of the R-tree index table (in megabytes).
- During R-tree index creation, users may require three times the size returned.
- Users or applications can control where the extra space is used via a new CREATE INDEX parameter:
  - 'Work_tablespace = work_indx_tblspc'

```
CREATE INDEX tablename_sidx
  ON tablename(geometry)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX
  PARAMETERS('work_tablespace = work_indx_tblspc');
```

ORACLE



### R-Tree Index Sizing: Usage Notes and WORK_TABLESPACE

The SDO_TUNE.ESTIMATE_RTREE_INDEX_SIZE function returns a number that is the estimate of the size of the R-tree index, given the set of input parameters. During index creation, up to three times the estimated size is needed for various temporary tables used during spatial index creation. Users or applications can control where that extra space is written to via the WORK_TABLESPACE spatial parameter of the CREATE INDEX command.

# Resources Required for Creating an R-Tree Spatial Index

- All geometries are read and MBRs are constructed.
  - Approximately 100 bytes of rollback is required per geometry.
- R-tree clustering operations are performed.
  - Requires a maximum of 1 GB temporary space for sorting (1 GB required for 5 million or more geometries)
  - Requires 10 MB of System Global Area (SGA) per index
- Use the R-tree index sizing function to estimate the final index size.

ORACLE





## Resources Required for Creating an R-Tree Spatial Index

An R-tree index is built in two stages. In the first phase, all the geometries are read and the MBRs associated with each geometry are created. In the second stage, the R-tree clustering operations are performed and the index table is created.

Each geometry requires 100 bytes of rollback segment.

During the second stage of the index build:

- The index build requires a maximum of 1 GB of temporary space for sorting operations. One gigabyte is required for 5 million or more geometries. If there are more than 5 million geometries, all sort operations operate on a maximum of 5 million geometry MBRs.
- Each index requires ten megabytes of SGA
- During R-tree index creation, you may require the tablespace size to be up to three times the final index size

# Summary

In this lesson, you should have learned how to:

- Explain the concept of spatial indexing
- Generate an R-tree spatial index
- Estimate the size of an R-tree index

ORACLE



## Summary

In this lesson, you learned about the concepts of spatial indexing.

# Practice 8: Overview

This practice covers creating nonspatial and R-tree spatial indexes on spatial layers.

ORACLE



## Practice 8: Overview

In this practice, you create nonspatial and R-tree spatial indexes on spatial layers.

## Practice 8

1. Create nonspatial indexes on the following columns. Make sure that all the indexes are created on the `indx_tblspc` tablespace that is already created for you.
   ```
   i. GEOD_CITIES(CITY)
   ii.GEOD_STATES(STATE)
   iii.GEOD_STATES(STATE_ABRV)
   iv.GEOD_INTERSTATES(HIGHWAY)
   v. GEOD_COUNTIES(STATE, COUNTY)
   vi.GEOD_COUNTIES(STATE_ABRV, COUNTY)
   ```
   **Note:** The following nonspatial indexes were created on the projected layers during import:
   ```
   i. PROJ_CITIES(CITY)
   ii.PROJ_STATES(STATE)
   iii.PROJ_STATES(STATE_ABRV)
   iv.PROJ_INTERSTATES(HIGHWAY)
   v. PROJ_COUNTIES(STATE, COUNTY)
   vi.PROJ_COUNTIES(STATE_ABRV, COUNTY)
   ```
2. Create spatial (R-tree) indexes on the following layers. Again, set the tablespace parameter to the `indx_tblspc` tablespace. Also, set the work tablespace parameter to the `work_indx_tblspc` tablespace.
   ```
   i. GEOD_CITIES(LOCATION)
   ii.GEOD_INTERSTATES(GEOM)
   iii.GEOD_COUNTIES(GEOM)
   iv.GEOD_STATES(GEOM)
   v. PROJ_CITIES(LOCATION)
   vi.PROJ_STATES(GEOM)
   vii.PROJ_INTERSTATES(GEOM)
   viii.PROJ_COUNTIES(GEOM)
   ```
3. Describe `USER_SDO_INDEX_METADATA`.
4. Describe the `USER_SDO_INDEX_INFO` view. Write a query to select all the rows from the `USER_SDO_INDEX_INFO` view.
5. Estimate the size of an R-tree index on a table that has 200,000 geometries, `DB_BLOCK_SIZE` set to 16 KB (16384), `SDO_RTR_PCTFREE` set to 10, two-dimensional data, and a geodetic `SRID`.
6. Run the `verify_indexes.sql` script to check whether all the required indexes were created. The script is located in the `D:\labs\labs` folder.

9

# Querying Spatial Data

ORACLE

# Objectives

After completing this lesson, you should be able to do the following:

- Explain the Oracle Spatial query model
- Describe and compare spatial operators and functions
- Describe the topological relationships used by the spatial operators and functions
- Use the `SDO_FILTER`, `SDO_RELATE`, and `SDO_<RELATIONSHIP>` operators
- Differentiate between the `SDO_RELATE` operator and the `SDO_GEOM.RELATE` function

ORACLE



## Objectives

This lesson introduces you to the concepts and examples of querying spatial data. You learn to use spatial operators and functions to query geometrical data.

# Lesson Agenda

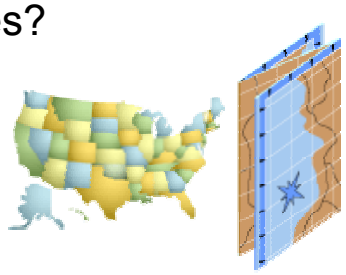
- Overview of the Spatial query model
- Overview of spatial operators, procedures, and functions
- `SDO_FILTER` operator
- Spatial topological relationships
- `SDO_RELATE` operator:
  - Geodetic optimized rectangle
  - `ORDERED` optimizer hint
  - Simplified relationship operators
  - Disjoint geometries query
  - Implicit coordinate system transformations
- `SDO_GEOM.RELATE` function:
  - `SDO_RELATE` or `SDO_GEOM.RELATE`

ORACLE

# Spatial Query Examples

Some examples of typical queries on spatial data:

- Which of the millions of roads in the U.S. have some interaction with the state of Texas?
- Which counties are inside the state of New Hampshire?
- What is the total population in a selected rectangular area of interest?
- Which interstates interact with Passaic County?
- What kind of topological relationships does the state of New Jersey have with its counties?



ORACLE

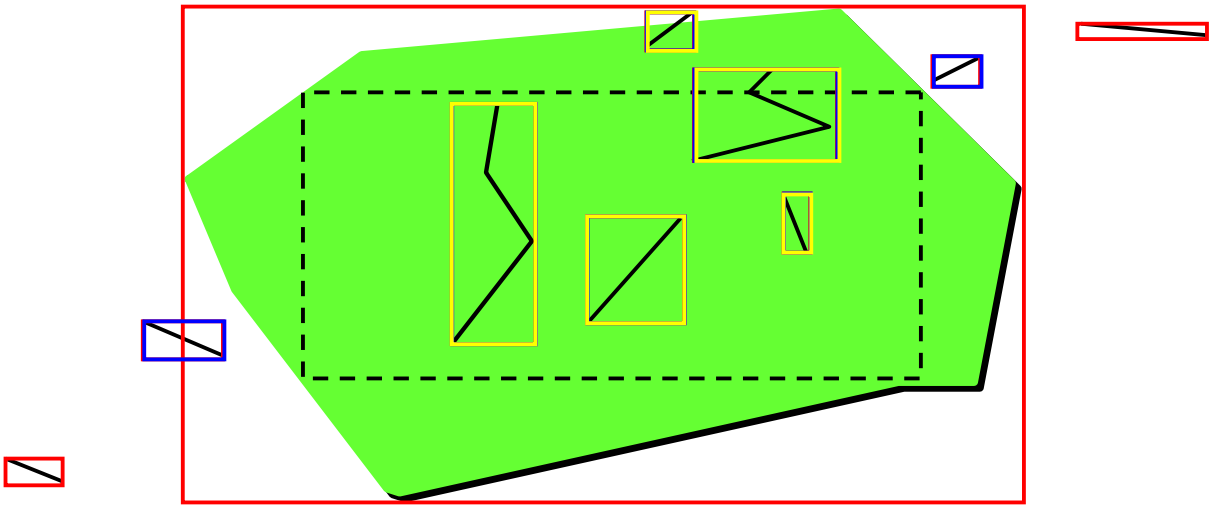


## Spatial Query Examples

In the slide are some typical queries that you may want answers to, from your spatial data. This lesson discusses the various spatial operators and functions that you can use to answer such queries precisely.

# Primary and Secondary Filters

- The primary filter compares geometry approximations, so the result is not exact.
- Interior optimizations are applied to the candidate set.
- Geometry comparisons are done only where required.



ORACLE



## Primary and Secondary Filters

Spatial uses a two-tier query model. The primary filter uses spatial indexes to compare geometry approximations and derives a superset of the result. The secondary filter may be applied on this superset to get the exact result set. Secondary filters use specific functions and operators to derive an exact result set.

In this example, the R-tree index is used to determine which geometries have any interaction with the rectangular geometry.

The R-tree spatial index is used to store approximations of every geometry stored. A minimum bounding rectangle (MBR) that covers each geometry is stored in the spatial index along with a pointer to that geometry. The query window also has an MBR. The spatial index works by comparing the approximation of the query window (MBR) with the approximations of geometries (MBRs in the index). All geometries with an index MBR that interacts with the query window MBR are returned by the spatial index query.

Note that the results of the index query (or primary filter) are not exact results because comparisons are done using approximations of each geometry. The spatial index always returns the correct final results of a spatial query, and perhaps some extra geometries.

## Primary and Secondary Filters (continued)

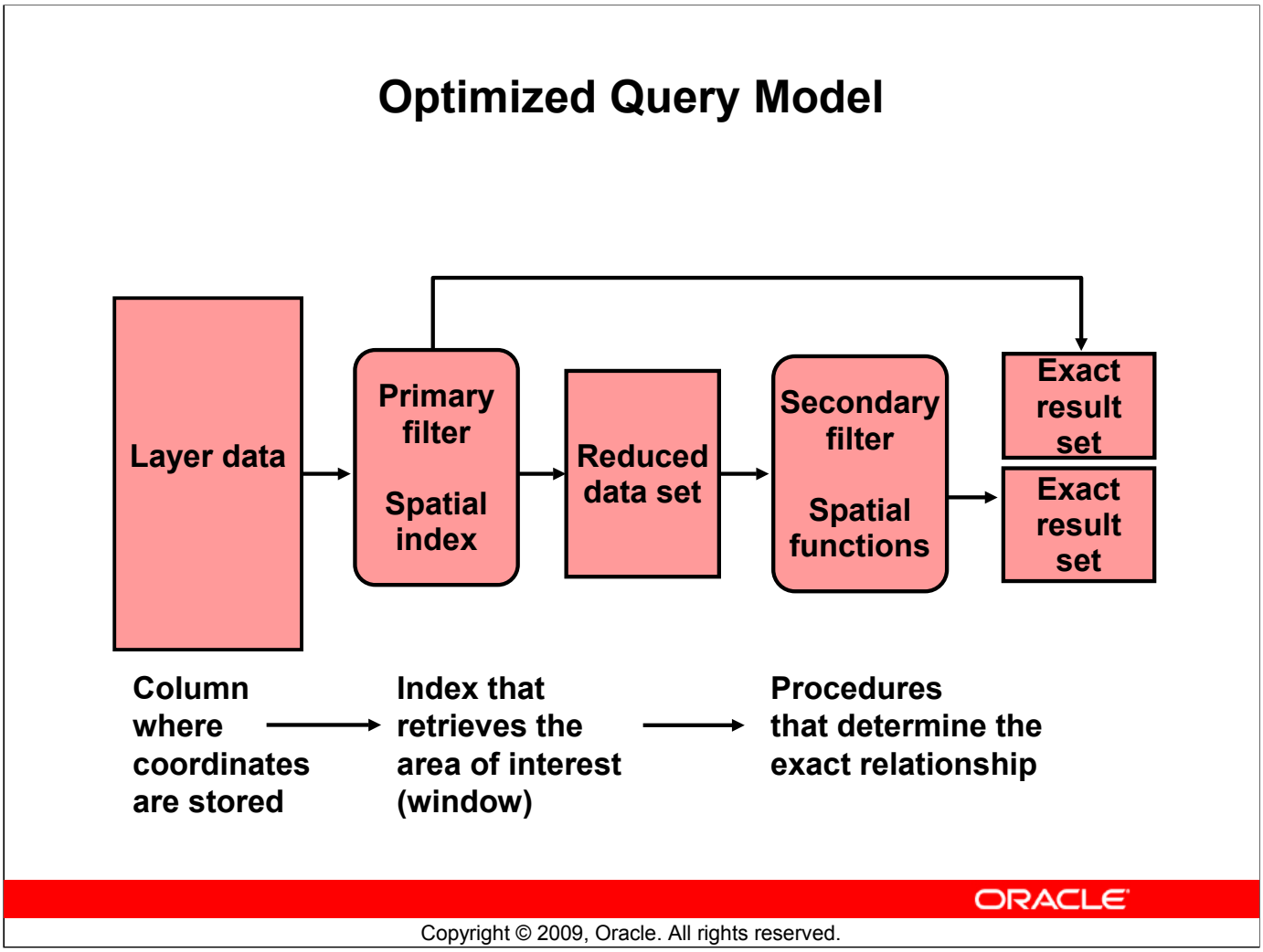
Next, interior optimizations are applied. The example shown is for illustration purposes only, and may not be exactly what Oracle Spatial does in all cases. In this case, an interior rectangle is generated, and is shown by the dotted line. Any geometry's MBR that falls completely in the interior rectangle of the county can be optimized into the result set with no true coordinate geometry calculations.

After all the optimizations are complete, Oracle Spatial performs true geometry-to-geometry comparisons, returning the last of the roads that have an interaction with the county.

A primary filter returns all the geometries that satisfy a query, and possibly some additional geometries. Some applications do not always need an exact answer. For example, the `ZOOM IN/OUT` functionality of a map display application can perform a primary filter in an Oracle database. It can then clip the results on the client to discard additional geometries returned from the primary filter.

Primary filters alone are not appropriate for every situation. For example, if you drew a circle on a viewport, and asked for all the parcels that interact with the circle, an approximate answer may not be acceptable. In this scenario, you can perform a secondary filter in the Oracle server to get the exact answer.

The secondary filter process first performs a primary filter. The primary filter can automatically accept a set of geometries as interacting, even though it compares only geometry approximations. Then the coordinates of the remaining geometries that are not automatically accepted in the primary filter are examined for possible interaction. The results of a secondary filter are all the geometries that interact, and nothing more.




## Optimized Query Model

This slide describes the Oracle Spatial optimized query model. It may not make much sense the first time you see it. This slide is repeated later on, and will make more sense then.

Start out with a spatial layer and ask a spatial question (for example, “What are all the parcels that interact with a given circular area?”).

Oracle Spatial has a two-stage query model: a primary filter and a secondary filter.

When a spatial index is created, geometry approximations are stored for each geometry. The primary filter portion of the query compares geometry approximations instead of true geometries, and generates a reduced data set. Comparing geometry approximations is much faster than comparing geometries.

# Lesson Agenda

- Overview of the Spatial query model
- Overview of spatial operators, procedures, and functions
- `SDO_FILTER` operator
- Spatial topological relationships
- `SDO_RELATE` operator:
  - Geodetic optimized rectangle
  - `ORDERED` optimizer hint
  - Simplified relationship operators
  - Disjoint geometries query
  - Implicit coordinate system transformations
- `SDO_GEOM.RELATE` function:
  - `SDO_RELATE` or `SDO_GEOM.RELATE`

ORACLE

# Spatial Operators

- Use a spatial index to provide optimum performance
- Require a spatial index on the first geometry specified in the operator
- Must be used in the `WHERE` clause of a query
- Include:
  - `SDO_FILTER`:
    - Performs only a primary filter
  - `SDO_RELATE` and `SDO_<RELATIONSHIP>`
    - Performs a primary and secondary filter

ORACLE



## Spatial Operators

Spatial operators take advantage of spatial indexes to provide optimum performance. The first parameter of any operator specifies the geometry column to be searched, and the second parameter specifies a query window. Spatial operators require that an index be built on the first geometry (the search column) specified in the operator. Spatial operators must be used only in the `WHERE` clause of a SQL statement.

Some spatial operators discussed in this lesson:

- **`SDO_FILTER`:** Uses the spatial index to compare geometry approximations in a spatial query to identify pairs of geometries that potentially interact. Geometries interact if they are not disjoint. Because `SDO_FILTER` compares only geometry approximations, it returns all the candidates that interact, and possibly a few more.
- **`SDO_RELATE`:** Uses the spatial index to identify geometries that have a specific type of interaction with one or more provided area-of-interest geometries. This operator performs both primary and secondary filter operations.

In the lesson titled "Using `SDO_WITHIN_DISTANCE`, `SDO_NN`, and `SDO_JOIN` Operators," you learn about `SDO_NN`, `SDO_WITHIN_DISTANCE`, and `SDO_JOIN` spatial operators.

# Spatial Operator Syntax Template

```
SDO_<operator_name>(
      <geometry-1>,
      <geometry-2> ) = 'TRUE'
```

- Spatial operators follow a template for its syntax.
- `<geometry-1>` is the column of the `SDO_GEOMETRY` type that you search. The column must be spatially indexed.
- `<geometry-2>` is the query window.
- With spatial operators, always specify `TRUE` in uppercase. Do not specify `<> 'FALSE'` or `= 'true'`.

ORACLE



## Spatial Operator Syntax Template

You can think of this spatial operator's syntax as a template. If you understand this spatial operator's syntax, you should easily understand the other spatial operators, all of which have a similar syntax.

The first geometry in the parameter list is the column that you are searching. The second geometry in the parameter list is the window (or area of interest). Optionally, you can specify a third parameter, which is a quoted string of parameters specific to the operator.

**<geometry-1>** is the column of the `SDO_GEOMETRY` type that you search. The column must be spatially indexed.

**<geometry-2>** is the query window.

**Note:** Some guidelines for the use of spatial operators:

- With operators, always specify `TRUE` in uppercase. Do not specify `<> 'FALSE'` or `= 'true'`.
- With operators, use the `/*+ ORDERED */` optimizer hint if the query window comes from a table. This is discussed later.

# Spatial Procedures and Functions

- Do not use spatial indexes
- Can be used on small tables that are not spatially indexed
- Can be used in the `SELECT` list or the `WHERE` clause
- Must be in the same coordinate system if two geometries are required
- Can be applied on the result set of a spatial operator

The spatial function discussed in this lesson:

`SDO_GEOM.RELATE`: To determine the relationship between two geometries without using a spatial index

ORACLE





## Spatial Procedures and Functions

Spatial procedures and functions are provided as subprograms in PL/SQL packages, such as `SDO_GEOM`, `SDO_CS`, and `SDO_UTIL`. Spatial procedures and functions do not take advantage of spatial indexes. They can be used on small tables that are not spatially indexed. But even if spatial columns are indexed, spatial functions do not use them. Spatial functions can be used both in the `SELECT` list and the `WHERE` clause of a SQL statement. If there are two geometries required as input to a spatial function, both must be in the same coordinate system. A spatial function does not implicitly convert the coordinate system of the query window to the coordinate system of the geometry being queried.

The spatial function discussed in this lesson:

`SDO_GEOM.RELATE`**:** Does not use a spatial index. It examines two geometry objects to determine their spatial relationship. This function is almost never used in the `WHERE` clause (the `SDO_RELATE` operator is used instead). `SDO_GEOM.RELATE` can be useful in the `SELECT` clause to report back the relationship between two geometries.

**Note:** Use spatial operators versus a spatial function if the operator satisfies your requirements. For example, use the `SDO_RELATE` operator rather than the `SDO_GEOM.RELATE` function.

# Lesson Agenda

- Overview of the Spatial query model
- Overview of spatial operators, procedures, and functions
- `SDO_FILTER` operator
- Spatial topological relationships
- `SDO_RELATE` operator:
  - Geodetic optimized rectangle
  - `ORDERED` optimizer hint
  - Simplified relationship operators
  - Disjoint geometries query
  - Implicit coordinate system transformations
- `SDO_GEOM.RELATE` function:
  - `SDO_RELATE` or `SDO_GEOM.RELATE`

ORACLE

# SDO_FILTER Operator

```
SDO_FILTER ( <geometry-1>,<geometry-2> ) = 'TRUE'
```

- **SDO_FILTER:**
  - Performs only primary filter operation
  - Uses the spatial index to identify the set of spatial objects that potentially interact with a query window
  - Must be used as an expression in the WHERE clause
  - Evaluates to TRUE for geometry approximation pairs that interact
- <geometry-1>: Is the search column
- <geometry-2>: Is the query window

ORACLE



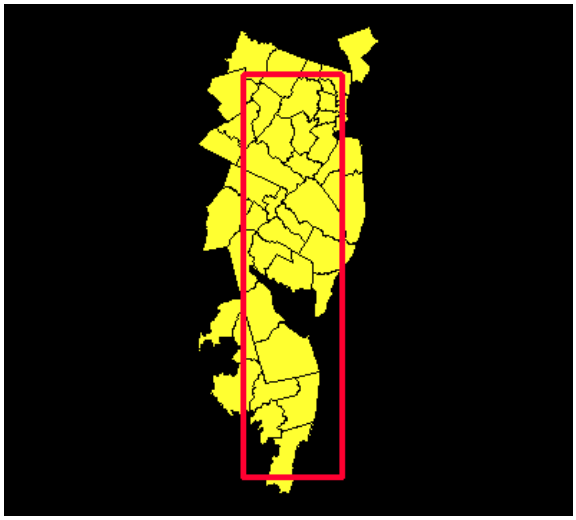
## SDO_FILTER Operator

The SDO_FILTER operator uses the spatial index to compare geometry approximations in a spatial query identifying pairs of geometries that potentially interact. Geometries interact if they are not disjoint. Because SDO_FILTER compares only geometry approximations, it returns all the candidates that interact, and possibly a few more. This operator performs only a primary filter operation.

The operator must always be used in a WHERE clause and the condition that includes the operator should be an expression of the form SDO_FILTER(arg1, arg2) = 'TRUE'.

## SDO_FILTER: Example

- Find all the counties that interact with a selected rectangular area.
- The result is approximate.



ORACLE



### SDO_FILTER: Example

The query given in the following slide finds all the counties that interact with a selected rectangular area, and returns an approximate result. The result is approximate because only a primary filter operation is being performed.

Note that the county in the upper-right corner does not interact with the rectangle, but is returned because its minimum bounding rectangle interacts with the rectangular query window.

## SDO_FILTER: Example

Find all counties whose approximations interact with a rectangular area:

```
SELECT c.county, c.totpop
FROM proj_counties c
WHERE sdo_filter (
        c.geom,
        sdo_geometry (2003, 32775, null,
          sdo_elem_info_array (1,1003,3),
          sdo_ordinate_array (1720300,1805461,
                              1820000, 2210000))
        ) = 'TRUE';
```

Note: All spatial operators return `TRUE` or `FALSE`. When writing spatial queries, always test with `= 'TRUE'`; never use `<> 'FALSE'` or `= 'true'`.

ORACLE



### SDO_FILTER: Example (continued)

This query finds all the counties likely to interact with a rectangular area, and returns an approximate result. The result is approximate because only a primary filter operation is being performed. The following is a brief description of the `SDO_FILTER` parameters:

- `<geometry-1>`: The search column is `C.GEOM` from the `PROJ_COUNTIES` table.
- `<geometry-2>`: The query window is created dynamically using the `SDO_GEOMETRY` constructor. It is a two-dimensional optimized rectangle defined by a pair of X,Y coordinates.

# Lesson Agenda

- Overview of the Spatial query model
- Overview of spatial operators, procedures, and functions
- `SDO_FILTER` operator
- Spatial topological relationships
- `SDO_RELATE` operator:
  - Geodetic optimized rectangle
  - `ORDERED` optimizer hint
  - Simplified relationship operators
  - Disjoint geometries query
  - Implicit coordinate system transformations
- `SDO_GEOM.RELATE` function:
  - `SDO_RELATE` or `SDO_GEOM.RELATE`

ORACLE

## Spatial Relationships with Secondary Filters

- To determine the spatial relationship between geometries, apply a secondary filter.
- Spatial relationships are based mainly on topology and distance.
- Each spatial object has a boundary, an exterior, and an interior.

ORACLE

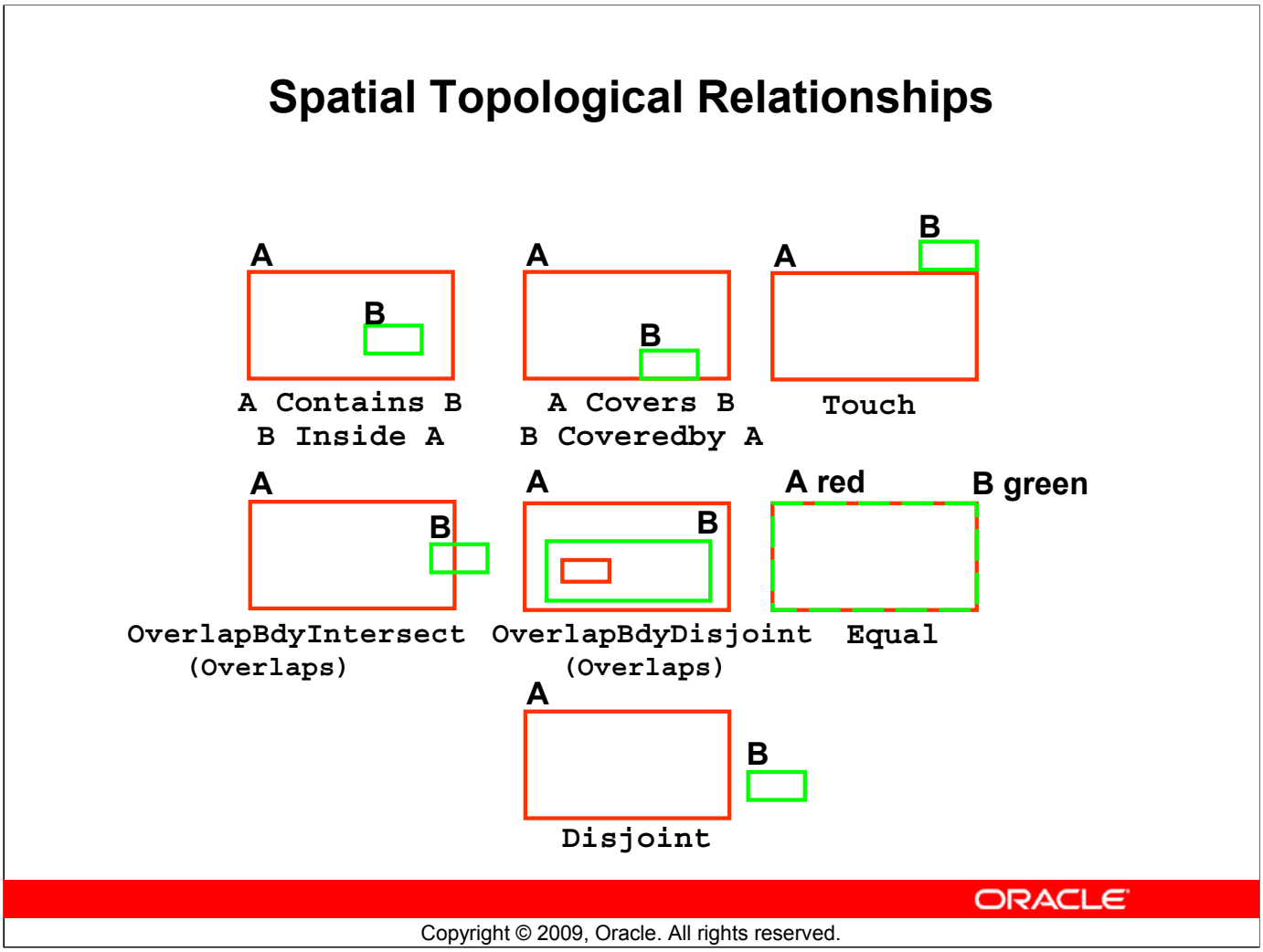


### Spatial Relationships with Secondary Filters

To determine the spatial relationship between geometries, you must use a secondary filter. The spatial relationship is based on geometry locations. Spatial relationships are based on topology and distance.

Each spatial object has an interior, a boundary, and an exterior. Points or lines that separate the interior of a geometry from the exterior are defined as its boundary. The boundary of a line string consists of its end points; however, if the end points overlap (that is, if they are the same point), the line string has no boundary.

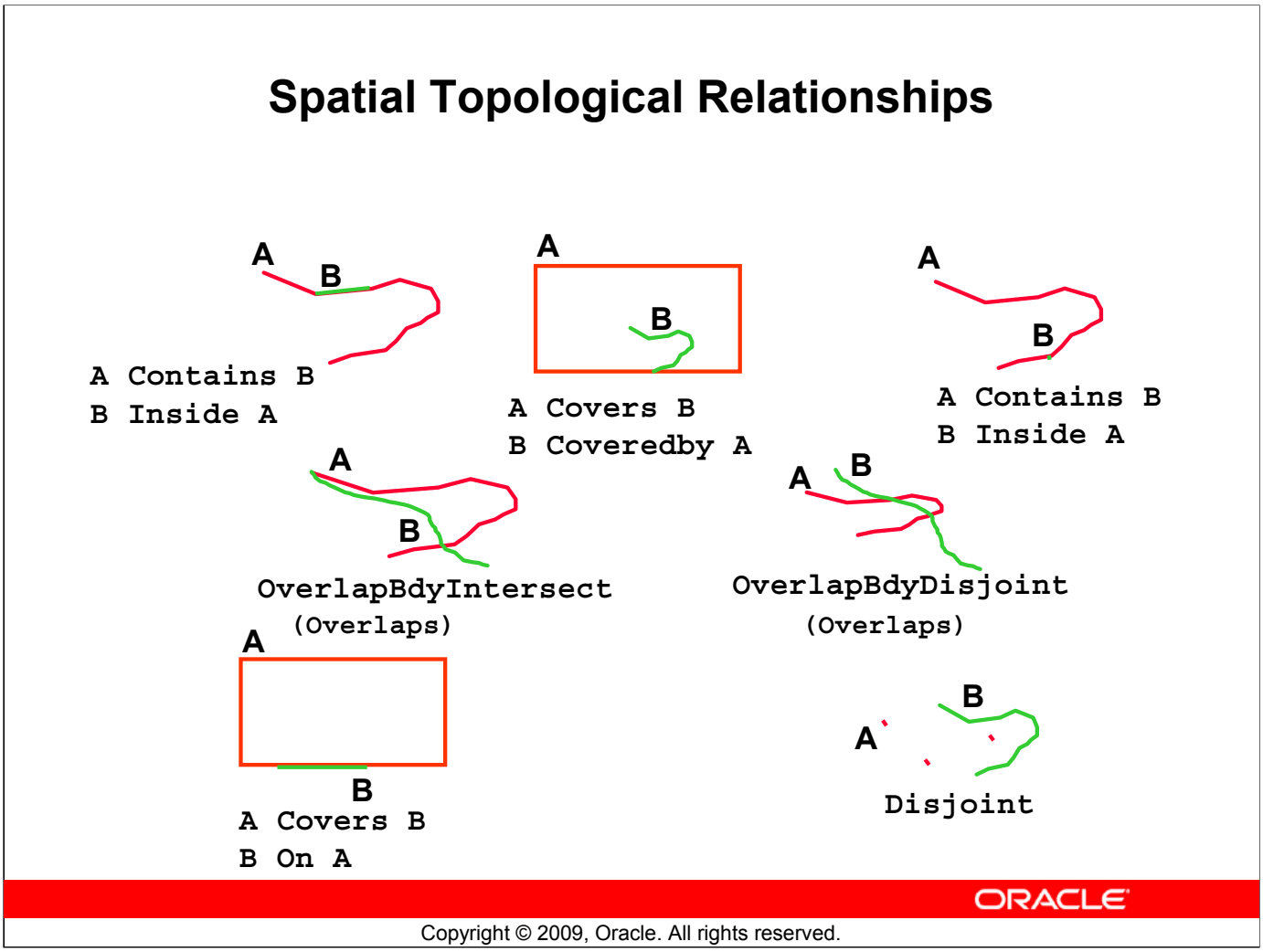
The perimeter of a polygon is the line that describes its boundary. The interior consists of points that are in the object but not on its boundary, and the exterior consists of those points that are not in the object.




## Spatial Topological Relationships

Some of the topological relationships identified in the seminal work by Professor Max Egenhofer (University of Maine, Orono) and colleagues have names associated with them. Oracle Spatial uses the following names:

- `CONTAINS`: The interior and boundary of one object are completely contained in the interior of the other object.
- `INSIDE`: This is the opposite of `CONTAINS`. A `INSIDE` B implies that B `CONTAINS` A.
- `COVERS`: The interior of one object is completely contained in the interior or boundary of the other object, and their boundaries intersect.
- `COVEREDBY`: This is the opposite of `COVERS`. A `COVEREDBY` B implies that B `COVERS` A.
- `TOUCH`: The boundaries intersect but the interiors do not intersect.
- `OVERLAPS`: Geometries are in the `OVERLAPBDYINTERSECT` or `OVERLAPBDYDISJOINT` topological relationship.
  - `OVERLAPBDYINTERSECT`: The boundaries and interiors of the two objects intersect.
  - `OVERLAPBDYDISJOINT`: The interior of one object intersects the boundary and interior of the other object, but the two boundaries do not intersect. This relationship occurs, for example, when a line originates outside a polygon and ends inside that polygon.
- `EQUAL`: The two objects have the same boundary and interior.




## Spatial Topological Relationships (continued)

- DISJOINT: The boundaries and interiors do not intersect.
- ON: The interior and boundary of the first object is on the boundary of the second object (and the second object covers the first object). This relationship occurs, for example, when a line is on the boundary of a polygon.

**Note:** The boundary of a line string is its end point.

# Spatial Topological Relationships

ANYINTERACT:

- Returns TRUE if geometries are not disjoint
- Is an optimal mask because it does not have to determine the relationship between geometries
- Only determines that the geometries are not disjoint

ORACLE



## Spatial Topological Relationships (continued)

- ANYINTERACT: The geometries are nondisjoint. This is an optimal mask because it does not have to determine geometry relationships. As soon as the algorithm determines that the geometries are not disjoint, it exits the algorithm.

# Lesson Agenda

- Overview of the Spatial query model
- Overview of spatial operators, procedures, and functions
- `SDO_FILTER` operator
- Spatial topological relationships
- `SDO_RELATE` operator:
  - Geodetic optimized rectangle
  - `ORDERED` optimizer hint
  - Simplified relationship operators
  - Disjoint geometries query
  - Implicit coordinate system transformations
- `SDO_GEOM.RELATE` function:
  - `SDO_RELATE` or `SDO_GEOM.RELATE`

ORACLE

## SDO_RELATE Operator

```
SDO_RELATE
 ( <geometry-1>,
   <geometry-2>,
   'MASK=<mask>'
 ) = 'TRUE'
```

- Performs both primary and secondary filters when processing a query
- Uses the secondary filter to ensure that only candidate objects that interact are selected
- Evaluates to TRUE for geometries that have the topological relationship specified by <mask>



### SDO_RELATE Operator

SDO_RELATE uses the spatial index to identify geometries that have a specified type of interaction with one or more area-of-interest geometries. This operator performs both primary and secondary filter operations.

The template for SDO_RELATE is similar to that for SDO_FILTER. The first parameter, <geometry-1>, is the column that you search. The second parameter, <geometry-2>, is the window (or area of interest). The third parameter is a quoted string of parameters specific to the operator.

The operator must always be used in a WHERE clause, and the condition that includes the operator should be an expression of the form SDO_RELATE(arg1, arg2, arg3) = 'TRUE'.

## SDO_RELATE: Arguments

- <geometry-1> is the layer to be searched.
  - Must be a column in a table
  - Must be of the SDO_GEOMETRY type
  - Must be indexed
- <geometry-2> is the query window.
  - Is a variable or column in a table
  - Must be of the SDO_GEOMETRY type
- <mask> identifies the spatial relationship to test.
  - Some valid mask keyword values are: CONTAINS, COVERS, ANYINTERACT, ON, and TOUCH
  - Multiple masks are combined as follows: 'INSIDE+COVEREDBY'

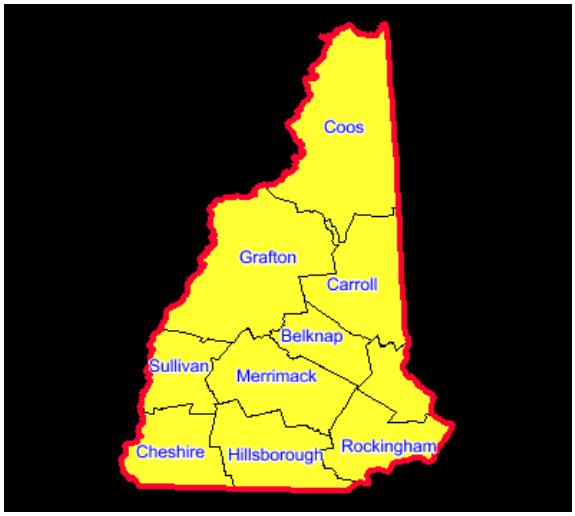
ORACLE



### SDO_RELATE: Arguments

- **<geometry-1>** is the spatial column of the SDO_GEOMETRY type of a table that is being searched. The column must be spatially indexed.
- **<geometry-2>** is either an SDO_GEOMETRY object from a table or a transient instance of a geometry specified using a bind variable or the SDO_GEOMETRY constructor. If the geometry2 column is not spatially indexed, the operator indexes the query window in memory for optimum performance.
  **Note:** If geometry1 and geometry2 are based on different coordinate systems, geometry2 is temporarily transformed to the coordinate system of geometry1.
- **<mask>** is used to specify the topological relation of interest. This is a required parameter. Valid values are one or more of the following in the nine-intersection pattern: TOUCH, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, EQUAL, INSIDE, COVEREDBY, CONTAINS, COVERS, ANYINTERACT, ON. Multiple masks are combined with the logical Boolean operator OR—for example, 'mask=INSIDE+TOUCH';.

# Find All Counties `INSIDE+COVEREDBY` the State of New Hampshire




ORACLE

## SDO_RELATE: Example

Find all counties in the state of New Hampshire:

```
SELECT c.county, c.state_abrv
FROM geod_counties c,
     geod_states s
WHERE s.state = 'New Hampshire'
 AND sdo_relate (c.geom,
                 s.geom,
                 'mask=INSIDE+COVEREDBY')
                    ='TRUE';
```

Note: For optimal performance, create a nonspatial index on `GEOD_STATES(state)`.

ORACLE



### SDO_RELATE: Example

This query finds all the counties in the state of New Hampshire (NH). It includes all the counties completely inside NH, as well as the counties inside NH that also touch its border.

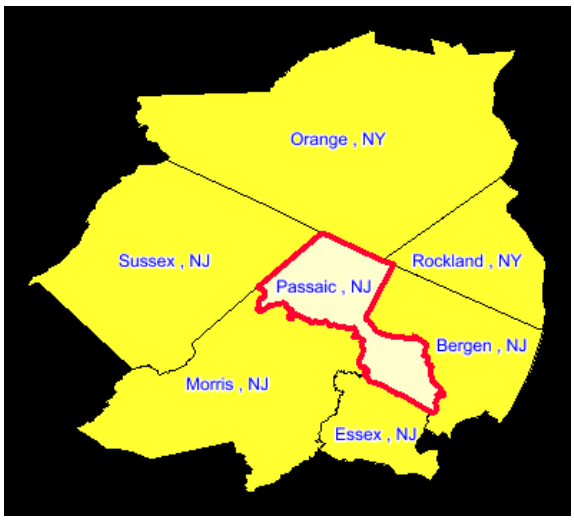
Because the `GEOD_COUNTIES` table has a `STATE_ABRV` column, this query could have been performed without Oracle Spatial. The query could have selected all the counties with `state_abrv = 'NH'`.

To perform this query with Oracle Spatial, you can use `SDO_RELATE`. `SDO_RELATE` performs a primary filter and a secondary filter and gives the exact result.

In the `WHERE` clause:

- `<geometry-1>` is the search column, which corresponds to `C.GEOM` from the `GEOD_COUNTIES` table
- `<geometry-2>` is the query window. The query window is `S.GEOM`, which corresponds to the state of New Hampshire.
  **Note:** In this query, the query window is a stored geometry. For optimal performance, create a nonspatial index on `GEOD_STATES(state)` because if the table is huge, it takes a lot of time to search and retrieve the query window.
- `mask=INSIDE+COVEREDBY` (ORed mask) specifies the spatial relationship that you want to test. The `SDO_RELATE` operator evaluates to `TRUE` for all the geometries representing counties that are inside or covered by the geometry of the state of New Hampshire.

# Find All Counties that TOUCH Passaic County in New Jersey




ORACLE

## SDO_RELATE: Example

Find all counties around Passaic County in New Jersey:

```
SELECT c1.county, c1.state_abrv
FROM geod_counties c1,
     geod_counties c2
WHERE c2.state = 'New Jersey'
  AND c2.county = 'Passaic'
  AND sdo_relate (c1.geom,
                  c2.geom,
                  'mask=TOUCH')='TRUE';
```

ORACLE



### SDO_RELATE: Example

This query finds all the counties that touch the border of Passaic County in New Jersey (NJ). SDO_RELATE uses a primary and a secondary filter and returns the exact result.

This query could not be done without Oracle Spatial. There are no attribute columns that you can query without Oracle Spatial to satisfy the TOUCH relationship.

**Note:** In this query, the GEOD_COUNTIES table is listed twice because the window (Passaic County in New Jersey) comes out of the same layer being searched.

In the WHERE clause:

- <geometry-1>: The search column is C1.GEOM from the GEOD_COUNTIES table.
- <geometry-2>: The window is C2.GEOM, Passaic County in New Jersey.
- mask=TOUCH: This specifies the spatial relationship that you want to test. The SDO_RELATE operator evaluates to TRUE for all the geometries representing counties that touch the geometry of the Passaic County in the state of New Jersey.

## SDO_RELATE: ANYINTERACT Example

Find all cities in a selected rectangular area:

```
SELECT c.city, c.pop90
FROM geod_cities c
WHERE sdo_relate (
       c.location,
       sdo_geometry (2003, 8307, null,
          sdo_elem_info_array (1,1003,3),
          sdo_ordinate_array (-109,37,-102,40)),
       'mask=ANYINTERACT') ='TRUE';
```

Note: For point-in-polygon queries, use the ANYINTERACT mask if you do not mind returning points that fall on the boundary; ANYINTERACT is a very fast SDO_RELATE operation.

Note: Because GEOD_CITIES contains only point data, for optimum performance, the index should have been created with LAYER_GTYPE=POINT.

ORACLE



### SDO_RELATE: ANYINTERACT Example

This query finds all the cities that have any interaction with the rectangular query window.

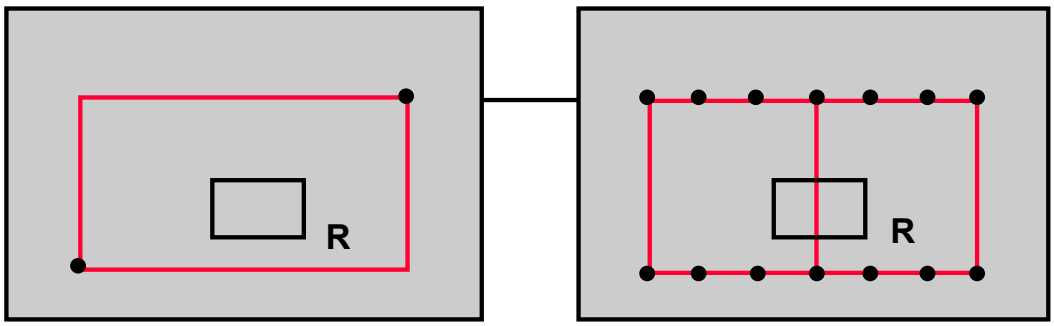
In the WHERE clause:

- <geometry-1>: The search column is C.LOCATION from the GEOD_CITIES table.
- <geometry-2>: The window is an optimized rectangle.
- mask=ANYINTERACT: This selects the geometries that have any interaction with a query window. The query window is an optimized rectangle.

**Note:** For point-in-polygon queries, use the ANYINTERACT mask if you do not mind returning points that fall on the boundary; ANYINTERACT is a very fast SDO_RELATE operation.

# Geodetic Optimized Rectangle

- Geodetic optimized rectangles may be internally represented as a multipolygon.
- Rectangle R is *inside* the first polygon; it is *not inside* the second multipolygon.




ORACLE



## Geodetic Optimized Rectangle

This is an example of a possible internal representation of a geodetic optimized rectangle.

Note that the internal representation may be a multipolygon whose edges touch. This is done to ensure that each of the polygon elements of the returned geometry has an area less than half the Earth's surface area.

Because the interior of the original optimized rectangle may get broken up into a geodetic multipolygon geometry, the new geometry must be used only for the following operations:

- As a window for `SDO_FILTER` queries
- As a window for `SDO_RELATE` queries, but only for the `ANYINTERACT` mask
- As a window for the `SDO_ANYINTERACT` operator, which is equivalent to `SDO_RELATE` with the `ANYINTERACT` mask

**Note:** All masks are allowed with a nongeodetic or projected optimized rectangle.

# Using SDO_RELATE in PL/SQL Code

Find the total population in a selected rectangular area:

```
set serveroutput on;
DECLARE
 rectangle         sdo_geometry;
 total_population number;
BEGIN
 rectangle := sdo_geometry (2003, 8307, null,
                sdo_elem_info_array (1,1003,3),
                sdo_ordinate_array (-109,37,-102,40));
 SELECT sum(c.totpop) into total_population
 FROM geod_counties c
 WHERE sdo_relate (c.geom, rectangle,
                   'mask=ANYINTERACT') = 'TRUE';
dbms_output.put_line('Population = '||
                        total_population||'.');
END;
/
```

ORACLE



## Using SDO_RELATE in PL/SQL Code

This query uses PL/SQL to find the total population of all counties that have any interaction with a selected rectangular area. The SDO_RELATE operator uses a primary and a secondary filter, and returns the exact result.

# Using the ORDERED Optimizer Hint

- For optimal performance, when the query window comes from a table, use the `/*+ ordered */` optimizer hint.
- The `ORDERED` hint instructs the optimizer to use indexes on tables in the order they are listed in the `FROM` clause.
- List the table for the query window first in the `FROM` clause.
- This enables Oracle Optimizer to find the query window first, and then search the spatial layer.





## Using the ORDERED Optimizer Hint

The `/*+ ordered */` hint is not a spatial hint. This is a generic Oracle hint that Oracle Spatial benefits from. The hint orders the driving tables in the query as they are listed in the `FROM` clause. The table containing `WINDOW` must always be listed first in the `FROM` clause.

Ensure that you use the `/*+ ordered */` hint for optimal performance when the query window comes from a table. A query window is always passed into the second argument of a spatial operator (that is, `SDO_FILTER`, `SDO_RELATE`, `SDO_WITHIN_DISTANCE`, or `SDO_NN`).

## SDO_RELATE: Example with the ORDERED Hint

Find all interstates that interact with a county:

```
SELECT /*+ ordered */ i.highway
FROM geod_counties c, geod_interstates i
WHERE c.state = 'New Jersey' and c.county = 'Passaic'
   AND sdo_relate (i.geom, c.geom,
       'mask=ANYINTERACT')='TRUE';
```

ORACLE



### SDO_RELATE: Example with the ORDERED Hint

This query finds all interstates that interact with Passaic County.

In the SDO_RELATE operator:

- <geometry-1>: The search column is i.geom from the geod_interstates table.
- <geometry-2>: The query window is c.geom, which is the Passaic County in New Jersey. This geometry is stored in the geod_counties table. The ORDERED optimizer hint is used to instruct the optimizer to first find the query window from the geod_counties table before beginning to search in the geod_interstates (geom) spatial layer. Note that the geod_counties table has been listed first in the FROM clause.

## SDO_RELATE: Example with Multiple Query Windows

Find all interstates that interact with selected counties:

```
SELECT /*+ ordered */ i.highway
FROM geod_counties c, geod_interstates i
WHERE c.state = 'Arizona' and c.poppsqmi < 10
   AND sdo_relate (i.geom, c.geom,
       'mask=ANYINTERACT')='TRUE';
```

ORACLE



### SDO_RELATE: Example with Multiple Query Windows

This query finds all interstates that interact with counties in the state of Arizona where the population per square mile is less than 10 people.

In the SDO_RELATE operator:

- <geometry-1>: The search column is i.geom from the GEOD_INTERSTATES table.
- <geometry-2>: The query window is c.geom, which includes all counties in Arizona with poppsqmi < 10. Because c.geom corresponds to all counties in Arizona with the population per square mile less than 10, it evaluates to more than one window. Therefore, for optimal performance, it is recommended that you use the ORDERED hint. Note that the geod_counties table has been listed first in the FROM clause.

# Simplified Relationship Operators

- Each SDO_RELATE mask value has a corresponding simplified relationship operator.
- The operator name is of the form SDO_<RELATIONSHIP> where <RELATIONSHIP> can be ANYINTERACT, INSIDE, COVERS, OVERLAPS, ON, COVEREDBY, and so on.

```
SDO_<RELATIONSHIP> (<geometry1>, <geometry2>) = 'TRUE'
```

- <geometry1>: Search column
- <geometry2>: Query window

ORACLE



## Simplified Relationship Operators

There are operators associated with each of the named topological relationships supported in Oracle Spatial. These operators are a simplified alternative to the SDO_RELATE operator.
<RELATIONSHIP> can take on any of the values passed in as a mask to SDO_RELATE:

```
SDO_<RELATIONSHIP> (<geometry-1>,<geometry-2>) = 'TRUE'
```

- <geometry-1> is the spatially indexed layer to be searched.
- <geometry-2> is the query window, which can be a variable or a column in a table of the SDO_GEOMETRY type.

Note that, apart from all SDO_RELATE masks, there is an additional relationship operator, SDO_OVERLAPS. This operator checks whether any geometries in a table overlap (that is, have the OVERLAPBDYDISJOINT or OVERLAPBDYINTERSECT topological relationship with) a specified geometry. This simplifies searches for overlapping relationships.

For more information about relationship operators, refer to the *Oracle Spatial Developer's Guide.*

# Relationship Operators: Example

- Find all counties around Passaic County in New Jersey:

```
SELECT /*+ ordered */ a.county
FROM geod_counties b,
     geod_counties a
WHERE b.county = 'Passaic'
    AND b.state = 'New Jersey'
    AND SDO_TOUCH(a.geom,b.geom) = 'TRUE';
```

- Previously:

```
. . .
AND SDO_RELATE(a.geom,b.geom,
     'MASK=TOUCH') = 'TRUE';
```

ORACLE



## Relationship Operators: Example

The example in the slide returns all counties that have a `TOUCH` relationship with Passaic County in New Jersey. The example uses the `SDO_TOUCH` operator: `SDO_TOUCH(a.geom,b.geom) = 'TRUE'`

The `SDO_TOUCH` operator is a simplified version of `SDO_RELATE` with the `TOUCH` mask: `SDO_RELATE(a.geom, b.geom, 'mask=touch') = 'TRUE'`

# Query to Get Disjoint Geometries

- There is no separate relationship operator or mask to get disjoint geometries.
- To get disjoint geometries, use the following query logic:

```
SELECT city, state_abrv
FROM geod_cities
  MINUS
SELECT /*+ ordered */ c.city, c.state_abrv
FROM geod_states s, geod_cities c
WHERE sdo_anyinteract(c.location,s.geom) = 'TRUE'
  AND s.state_abrv='CA';
```

- From all geometries, `MINUS` the geometries that have any interaction.
- Geometries that do not interact or are disjoint are returned.

ORACLE



## Query to Get Disjoint Geometries

Spatial operators always use the spatial index, which compares geometry approximations and returns geometries that are likely to interact. When using `SDO_RELATE` or relationship operators, secondary filters are applied. There is no specific mask or a relationship operator that returns geometries that are disjoint. Therefore, you must apply a different query logic to get disjoint geometries.

To get disjoint geometries, use a query similar to the one shown in the slide.

# Implicit Coordinate System Transformations

- In a spatial operator, the query window geometry can be in a different coordinate system from the geometry layer being searched.
- The query window is automatically transformed to the coordinate system of the layer.

ORACLE



## Implicit Coordinate System Transformations

Oracle Spatial operators perform implicit coordinate system transformations, if required. An implicit coordinate system transformation is required if the query window (`<geometry-2>`) is in a different coordinate system from the layer being searched (`<geometry-1>` in all operators). When an implicit coordinate system transformation happens, the query window geometry is automatically transformed to the coordinate system of the layer.

When a spatial operator is run, if `SDO_SRID` of the window geometry does not match `SDO_SRID` of the layer being searched, Oracle Spatial implicitly transforms the window geometry to the same coordinate system (based on the `SDO_SRID` value) as the layer being searched.

# Implicit Coordinate System Transformations: Example

- Find all counties in the state of New Hampshire.
  - The state of New Hampshire is in a projected coordinate system.
  - The counties are in a geodetic coordinate system.

```
SELECT /*+ ordered */ c.county, c.state_abrv
FROM proj_state s,
     geod_counties c
WHERE s.state = 'New Hampshire'
 AND sdo_inside (c.geom,s.geom)='TRUE';
```

- `s.geom` is implicitly transformed to the coordinate system of the `c.geom` layer.

ORACLE



### Implicit Coordinate System Transformations: Example

This query finds all counties that have any interaction with the state of New Hampshire (NH). It includes all counties completely inside NH, as well as counties inside NH that also touch its border. Therefore, the `SDO_INSIDE` operator is used.

Because the query window (the state of New Hampshire) and the layer are in different coordinate systems, Oracle Spatial automatically changes the coordinate system of the state of New Hampshire to the coordinate system of the `GEOD_COUNTIES(GEOM)` layer.

# Lesson Agenda

- Overview of the Spatial query model
- Overview of spatial operators, procedures, and functions
- `SDO_FILTER` operator
- Spatial topological relationships:
- `SDO_RELATE` operator
  - Geodetic optimized rectangle
  - `ORDERED` optimizer hint
  - Simplified relationship operators
  - Disjoint geometries query
  - Implicit coordinate system transformations
- `SDO_GEOM.RELATE` function:
  - `SDO_RELATE` or `SDO_GEOM.RELATE`

ORACLE

## SDO_GEOM.RELATE Function

```
SDO_GEOM.RELATE
 ( <geometry-1>, '<mask>', <geometry-2>, <tolerance> )
  = '<relationship>'
```

- Examines two geometry objects to determine their spatial relationship
- Performs an exact query (secondary filter)
- Returns TRUE or FALSE for an ANYINTERACT mask
- Returns the matching relationship if any other mask is used; returns FALSE if the relationship specified in the mask does not exist between the geometries
- Returns the name of the relationship if the DETERMINE mask is used
- Can be used in the SELECT list

ORACLE



### SDO_GEOM.RELATE Function

The SDO_GEOM.RELATE function examines two geometries to determine their spatial relationship. It returns different values depending on what mask you pass as an argument.

- If you pass a mask listing one or more relationships, it returns the name of the relationship if it is true for the pair of geometries. If all relationships are false, the procedure returns FALSE.
- If you pass the DETERMINE keyword in the mask, the function returns one relationship keyword that best matches the geometries.
- If you pass the ANYINTERACT keyword in the mask, the function returns TRUE if the two geometries are not disjoint.

All spatial functions, including SDO_GEOM.RELATE, do not use a spatial index. This function is almost never used in the WHERE clause. SDO_GEOM.RELATE can be useful in the SELECT clause to report back the relationship between two geometries. The following *mask* relationships can be tested: ANYINTERACT, CONTAINS, COVEREDBY, COVERS, DISJOINT, EQUAL, INSIDE, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, TOUCH, and DISJOINT (DISJOINT is returned if the objects have no common boundary or interior points).

Use the DETERMINE mask to find out the relationship between two geometries. Also, because it is a spatial function, it does not perform implicit coordinate system conversion. An exception is raised if geom1 and geom2 are based on different coordinate systems.

## SDO_GEOM.RELATE: Example with the DETERMINE Mask

- Determine the relationship of the state of New Jersey to its counties:

```
SELECT c.county,
       sdo_geom.relate (s.geom,'determine',c.geom, 0.5)
       relationship
FROM geod_states s,
     geod_counties c
WHERE s.state = 'New Jersey'
  AND s.state = c.state;
```

- The query returns:

```
COUNTY                         RELATIONSHIP
------------------------------ --------------
Atlantic                       COVERS
Cape May                       COVERS
Cumberland                     COVERS
Essex                          CONTAINS
```

ORACLE



### SDO_GEOM.RELATE: Example with the DETERMINE Mask

The DETERMINE mask keyword does not apply with the SDO_RELATE operator. It can be used to get the relationship between two geometries. This example determines the relationship of the state of New Jersey to its counties. It does not use a spatial index.

- New Jersey COVERS Atlantic County.
- New Jersey COVERS Cape May County.
- New Jersey COVERS Cumberland County.
- New Jersey CONTAINS Essex County.

## SDO_GEOM.RELATE: Example with the DETERMINE Mask

- Determine the relationship of New Jersey's counties to the state of New Jersey:

```
SELECT c.county,
       sdo_geom.relate(c.geom,'determine',s.geom, 0.5)
       relationship
FROM geod_states s,
     geod_counties c
WHERE s.state = 'New Jersey'
AND s.state = c.state;
```

- The query returns:

```
COUNTY                         RELATIONSHIP
------------------------------ --------------
Atlantic                       COVEREDBY
Cape May                       COVEREDBY
Cumberland                     COVEREDBY
Essex                          INSIDE
```

ORACLE



### SDO_GEOM.RELATE: Example with the DETERMINE Mask (continued)

This example determines the relationship of New Jersey's counties to the state of New Jersey. This is the inverse relationship of the one seen in the previous slide. It does not use a spatial index.

- Atlantic County is COVEREDBY New Jersey.
- Cape May County is COVEREDBY New Jersey.
- Cumberland County is COVEREDBY New Jersey.
- Essex County is INSIDE New Jersey.

## SDO_GEOM.RELATE: Example with the TOUCH Mask

- Find all counties around New Jersey:

```
SELECT c.county, c.state
FROM geod_states s,
     geod_counties c
WHERE s.state = 'New Jersey'
  AND sdo_geom.relate(c.geom,'touch',s.geom, 0.5)
      ='TOUCH';
```

- The function does not take advantage of a spatial index.
  - Use SDO_RELATE (or any of the SDO_<RELATIONSHIP> operators) instead.

ORACLE



### SDO_GEOM.RELATE: Example with the TOUCH Mask

This query finds all the counties around the state of New Jersey. It does not use a spatial index. SDO_GEOM.RELATE is almost never called in the WHERE clause of a SQL statement because it does not take advantage of a spatial index. Use SDO_RELATE (or any of the SDO_* simplified relationship operators) instead.

## SDO_RELATE or SDO_GEOM.RELATE

Use SDO_RELATE:

- In most cases, because a spatial index is more selective than any other index

Use SDO_GEOM.RELATE:

- When a nonspatial index is more selective than the spatial index
- When comparing just a few geometries

ORACLE



### SDO_RELATE or SDO_GEOM.RELATE

In most cases, a spatial index is more selective than any other index. In such cases, the SDO_RELATE operator must be used.

Sometimes, a nonspatial index is much more selective than a spatial index. When a nonspatial index is much more selective, use the SDO_GEOM.RELATE function instead of a spatial operator (SDO_RELATE or SDO_<RELATIONSHIP> operators such as SDO_ANYINTERACT and SDO_INSIDE).

Both the DISJOINT and DETERMINE masks provided by the SDO_GEOM.RELATE function are not supported by the SDO_RELATE operator.

## SDO_GEOM.RELATE: Example

Consider the following example:

- Data: The US_IMAGERY table has 10 million satellite images.
  - Nonspatial index on the CLOUD_COVER percentage
  - Nonspatial index on WAVELENGTH
  - Spatial index on GEOM, which is the extent of each image
- Query: Find all the imagery in the eastern United States with CLOUD_COVER percentage = 0 (very clear day) and WAVELENGTH = 400.

ORACLE



### SDO_GEOM.RELATE: Example

Consider the following example: A table called US_IMAGERY has 10 million satellite images. Some nonspatial columns (WAVELENGTH, CLOUD_COVER percentage) have traditional B-tree indexes. The GEOM column represents the spatial extent of each image, and is spatially indexed.

What is the best way to find all the imagery in the eastern United States with CLOUD_COVER percentage = 0 (very clear day) and WAVELENGTH = 400?

## SDO_GEOM.RELATE: Example

- What if the combination of CLOUD_COVER and WAVELENGTH predicates are much more selective than the spatial predicate (eastern United States)?
  - The spatial predicate alone returns five million rows.
  - Nonspatial predicates alone return 15 rows.
- Answer: Use SDO_GEOM.RELATE.

```
SELECT i.image_id
FROM us_areas a,
     us_images i
WHERE i.cloud_cover = 0
  AND i.wavelength = 400
  AND a.area = 'Eastern US'
  AND sdo_geom.relate(i.geom, 'anyinteract', a.geom,  0.5)
      ='TRUE';
```

ORACLE



### SDO_GEOM.RELATE: Example (continued)

What if the nonspatial predicates (WAVELENGTH and CLOUD_COVER) are much more selective than the spatial predicate? For example:

- If the spatial predicate were applied (without the other predicates), suppose five million rows return. This is not very selective, and at least five million geometry pairs are compared.
- If nonspatial predicates were applied (without the spatial predicate), suppose 15 rows return.

For this example, it is better not to use the SDO_RELATE (or SDO_<RELATIONSHIP> simplified relationship operators such as SDO_ANYINTERACT).

Instead, apply only nonspatial predicate indexes, which return 15 rows. SDO_GEOM.RELATE in the WHERE clause compares 15 geometry pairs to ensure that the final results set interacts with the query window. In this example, eastern United States is not a very selective spatial predicate. You use nonspatial predicate indexes first (on CLOUD_COVER and WAVELENGTH). Then apply SDO_GEOM.RELATE to ensure that the result includes images in the eastern part of the United States only.

# Summary

In this lesson, you should have learned how to:

- Explain the Oracle Spatial query model
- Describe and compare spatial operators and functions
- Describe the topological relationships used by the spatial operators and functions
- Use the `SDO_FILTER`, `SDO_RELATE`, and `SDO_<RELATIONSHIP>` operators
- Differentiate between the `SDO_RELATE` operator and the `SDO_GEOM.RELATE` function

ORACLE



## Summary

In this lesson, you learned about the concepts of the Spatial query model. You learned how to use different spatial operators and functions to solve spatial queries.

# Practice 9: Overview

- This practice covers the following topics:
  - Running primary filter queries
  - Using MapViewer to view query results
  - Running geodetic optimized rectangle queries
- Data sets loaded so far:
  - Geodetic `SRID` = 8307 (Longitude/Latitude [WGS 84]):
    - `GEOD_STATES`, `GEOD_COUNTIES`, `GEOD_INTERSTATES`, `GEOD_CITIES`
  - Projected `SRID` = 32775 (Equal-Area Projection [United States]):
    - `PROJ_STATES`, `PROJ_COUNTIES`, `PROJ_INTERSTATES`, `PROJ_CITIES`

ORACLE



## Practice 9: Overview

In this practice, you run spatial queries that apply some spatial operators such as `SDO_FILTER` and `SDO_ANYINTERACT`. You view the output of some of the queries in the MapViewer Simple Spatial Query Visualizer. You also run a query that uses the `SDO_GEOM.RELATE` function.

So far, you have loaded eight layers. Four of the layers are defined using a geodetic coordinate system:

- `GEOD_STATES`
- `GEOD_COUNTIES`
- `GEOD_INTERSTATES`
- `GEOD_CITIES`

Four of the layers are in a projected coordinate system:

- `PROJ_STATES`
- `PROJ_COUNTIES`
- `PROJ_INTERSTATES`
- `PROJ_CITIES`

Using the `GEOD_*` and `PROJ_*` prefixes, you can recognize whether a query against a geodetic layer or a projected layer is being demonstrated.

## Practice 9

1. From SQL*Plus, perform a primary filter query to find all counties that are likely to interact with the state of New York. Use the `geod_states` and `geod_counties` tables.
2. This exercise requires that OC4J is running and that a connection is properly defined to use MapViewer. If OC4J is not running, follow the directions in Practice 7 to start OC4J and define a connection to the database. Use the MapViewer Simple Spatial Query Visualizer to see the state of New York, and all counties that are likely to interact with it.
3. Find all counties that have an interaction with the state of New York.
4. Modify the query that you wrote for question 2 to display only the counties that have an interaction with the state of New York.
5. Find all cities and the population of the cities in the rectangle that has a lower-left coordinate of (–109, 37) and an upper-right coordinate of (–102, 40). Use the `geod_cities` table.
6. Find all counties and the population of each county that interact with the rectangle that has a lower-left coordinate of (–109, 37) and an upper-right coordinate of (–102, 40).
7. Determine the relationship between the state of New Hampshire and its counties. Set `COL` relation to `A20`.

10

# Using the SDO_WITHIN_DISTANCE, SDO_NN, and SDO_JOIN Operators

# Objectives

After completing this lesson, you should be able to use the following:

- The `SDO_WITHIN_DISTANCE` operator to solve distance-related queries
- The `SDO_NN` operator to perform the nearest-neighbor query
- The `SDO_JOIN` operator to get a spatial cross-product

ORACLE



## Objectives

This lesson provides information about some important spatial operators and functions. In this lesson, you learn to use the `SDO_WITHIN_DISTANCE` operator to solve distance-related queries. You learn to query spatial data for the nearest-neighbor geometries by using the `SDO_NN` operator and also to join geometries from different layers by using the `SDO_JOIN` operator.

# Lesson Agenda

- Spatial queries and operators
- SDO_WITHIN_DISTANCE operator
- SDO_NN operator
  - SDO_NN_DISTANCE ancillary operator
- Spatial join by using the SDO_JOIN operator
- Available features in Oracle Locator

# Spatial Query Examples

- Find all the cities within a distance from an interstate.
- Find five cities nearest to Interstate I170.
- Find all the city-and-county pairs that have any interaction.
- Find all cities within 10 miles of all interstate highways.

ORACLE



## Spatial Query Examples

In the slide are some of the typical spatial queries that you may want answers to. This lesson discusses the various spatial operators and functions that you can use to answer such queries precisely.

# Spatial Operators

- Spatial operators that can be used to solve distance- or proximity-related queries:
  - `SDO_WITHIN_DISTANCE`
  - `SDO_NN`
- The `SDO_JOIN` spatial operator:
  - Can be used to compare all geometries from one layer with all geometries in another layer
  - Is useful when comparing most of one layer with all or most of another layer

ORACLE



## Spatial Operators

The spatial operators discussed in this lesson help in solving such queries as listed in the previous slide. For distance, proximity, or nearest-neighbor queries, you can use the `SDO_WITHIN_DISTANCE` or `SDO_NN` operator.

# Lesson Agenda

- Spatial queries and operators
- SDO_WITHIN_DISTANCE operator
- SDO_NN operator
  - SDO_NN_DISTANCE ancillary operator
- Spatial join by using the SDO_JOIN operator
- Available features in Oracle Locator

ORACLE

## SDO_WITHIN_DISTANCE Operator

```
SDO_WITHIN_DISTANCE
 ( <geometry-1>,
   <geometry-2>,
   'DISTANCE=<n>,
   [parameters]'
 ) = 'TRUE'
```

- **SDO_WITHIN_DISTANCE**: Is used to determine the set of objects in a table that are within *n* distance units from a reference object (query window)
- <geometry1>: Is the search column
- <geometry2>: Is the query window
- <DISTANCE=n>: Is expressed in the units used for the coordinate system
- <unit>: Is the unit of measurement to associate with the <DISTANCE> parameter

ORACLE



### SDO_WITHIN_DISTANCE Operator

The SDO_WITHIN_DISTANCE operator uses the spatial index to identify the set of geometries that are within some specified distance of a given geometry (such as an area or point of interest). This operator returns the exact result by default, or can be made to return an approximate result.

The template for SDO_WITHIN_DISTANCE is similar to all other operators. The first parameter, geometry1, is the column that you are searching. The second parameter, geometry2, is the window (or area of interest) that you specify a distance from. The third parameter is a quoted string of parameters that are specific to the operator.

The SDO_WITHIN_DISTANCE(arg1, arg2, arg3) = 'TRUE' expression returns TRUE for all geometries in the search column (arg1) that are within the specified distance from the window (arg2). Otherwise, it returns FALSE.

The distance between two nonpoint objects, such as lines and polygons, is defined as the minimum distance between these two objects. The distance between two adjacent polygons is zero. The search column must have a spatial index built on it. The operator must always be in a WHERE clause.

The operator must always be used in a WHERE clause and the condition that includes the operator must be an expression of the form:

```
SDO_WITHIN_DISTANCE(arg1,arg2, 'distance = <value>') = 'TRUE'
```

## SDO_WITHIN_DISTANCE Operator (continued)

The arguments accepted by the operator are described in the following list:

- <geometry-1>: Is the spatial column that is being searched. It specifies an SDO_GEOMETRY column in a table that must be spatially indexed.
- <geometry-2>: Is the query window. This is the geometry that is buffered by distance.
- <distance>: Specifies the distance value. This is a required parameter whose data type is NUMBER. It can be associated with a UNIT. The third argument of this operator is specified as a quoted string that may include the DISTANCE parameter along with some optional parameter.

Apart from the DISTANCE parameter, you can specify additional optional parameters to control the behavior of the operator. Some important ones are discussed:

- <querytype>: Set querytype = FILTER to perform a primary filter operation only. If querytype is not specified, both primary and secondary filter operations are performed. The data type is VARCHAR2.
- <unit>: Specify the unit of measurement; a quoted string with unit= and an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table (for example, 'unit=KM').

**Note:** If UNIT is not specified, it defaults to the following:

- For geodetic data, the default unit is meters.
- For projected data, the default unit is the coordinate system unit.

## SDO_WITHIN_DISTANCE: Examples

- Find all cities within a distance from an interstate:

```
SELECT /*+ ordered */ c.city
FROM geod_interstates i, geod_cities c
WHERE i.highway = 'I170'
   AND sdo_within_distance (
        c.location, i.geom,
       'distance=15 unit=mile') = 'TRUE';
```

- Find interstates within a distance from a city:

```
SELECT /*+ ordered */ i.highway
FROM geod_cities c, geod_interstates i
WHERE c.city = 'Tampa'
   AND sdo_within_distance (
        i.geom, c.location,
        'distance=15 unit=mile') = 'TRUE';
```

ORACLE



### SDO_WITHIN_DISTANCE: Examples

The first query finds all the cities within 15 miles from an interstate; by default, the result is exact because both primary and secondary filter operations are performed.

In SDO_WITHIN_DISTANCE, the arguments are:

- <geometry1>: Is the search column, which is C.LOCATION from the GEOD_CITIES table
- <geometry2>: Is the query window, I.GEOM, which is I170 from the GEOD_INTERSTATES table
- 'distance=15 unit=mile': Specifies the distance in miles

Similarly, the second query finds all the interstates within 15 miles from a city named "Tampa."

# Lesson Agenda

- Spatial queries and operators
- SDO_WITHIN_DISTANCE operator
- SDO_NN operator
  - SDO_NN_DISTANCE ancillary operator
- Spatial join by using the SDO_JOIN operator
- Available features in Oracle Locator

# Nearest-Neighbor Operator: SDO_NN

```
SDO_NN
 ( <geometry-1>, <geometry-2>
   [,'parameters'][,tag]
 ) = 'TRUE'
```

- **SDO_NN**:
  - Uses the spatial index to identify the nearest neighbors for a geometry
  - Can specify the number of nearest neighbors to be returned
  - Does not return results in order of distance
  - Has an ancillary operator, SDO_NN_DISTANCE, that returns the distance associated with a nearest neighbor
- <geometry1>: Is the spatial column being searched
- <geometry2>: Is the geometry whose nearest neighbors you are looking for

ORACLE





## Nearest-Neighbor Operator: SDO_NN

This operator determines the nearest-neighbor geometries to a geometry. By default, it returns one nearest neighbor. To determine how near two geometry objects are, the shortest possible distance between the boundaries of each object is used. You can specify the number of the nearest neighbors to be returned. By default, it returns one nearest neighbor. The returned geometries are not in any order, unless you order by distance.

- <geometry1>: Is the spatial column being searched for the nearest neighbors. It specifies an SDO_GEOMETRY column in a table. The column must be spatially indexed.
- <geometry-2>: Is the geometry whose nearest neighbors you are looking for. It specifies either a geometry from a table or a transient instance of a geometry that can be specified using a bind variable or the SDO_GEOMETRY constructor.
- 'parameters': Defines the behavior of the operator. The parameters are discussed in the next few slides.
- SDO_NN_DISTANCE: Is an ancillary operator to the SDO_NN operator. It returns the distance of an object returned by the SDO_NN operator and is valid only within a call to the SDO_NN operator. The SDO_NN_DISTANCE operator is described in the following slides.

## SDO_NN: Optional Arguments

- DISTANCE: Is the optional maximum distance to consider when searching for the nearest neighbors
- SDO_NUM_RES: Defines the number of nearest neighbors to return. It takes only proximity into account.
- SDO_BATCH_SIZE: Specifies the number of rows to be processed at a time when evaluating the WHERE clause for other constraints
  - By default, set to 0 in which case Spatial decides the optimal batch size.
  - Use the ROWNUM pseudocolumn to limit the number of geometries returned.
- Specify either SDO_BATCH_SIZE or SDO_NUM_RES, but never both.
  - If neither is specified, the default is SDO_BATCH_SIZE=0.

ORACLE



### SDO_NN: Optional Arguments

DISTANCE: Specifies the number of distance units after which to stop searching for the nearest neighbors. If you do not also specify the unit keyword, the default is the unit of measurement associated with the data, for example:

```
'distance=10 unit=mile'
```

SDO_NUM_RES: Specifies the number of the nearest neighbors to return. SDO_NUM_RES takes only proximity into account. When you use the SDO_NUM_RES parameter, no other constraints are used in the WHERE clause.

SDO_BATCH_SIZE: Specifies the number of rows to be processed at a time when evaluating the WHERE clause for other constraints. SDO_BATCH_SIZE, or the number of rows, is continuously returned until all the constraints in the WHERE clause are satisfied. This is a performance-tuning parameter. You must choose an SDO_BATCH_SIZE parameter that minimizes the number of distances that SDO_NN calculates internally to satisfy your query. If SDO_BATCH_SIZE=0, SDO_NN automatically tunes the batch size depending on the result set size, adjusting it during query execution.

**Note:** Specify either SDO_BATCH_SIZE or SDO_NUM_RES, but never both. If neither is specified, the default is SDO_BATCH_SIZE=0.

## SDO_NN: Optional Arguments (continued)

**Note**

- If you specify the optional SDO_NUM_RES parameter, you can ask for the number of nearest neighbors you want, but no other condition in the WHERE clause is evaluated. For example, you want the five closest banks from an intersection, but only with the bank name 'CHASE'. If the five closest banks are not named 'CHASE', SDO_NN with SDO_NUM_RES=5 returns no rows. SDO_NUM_RES takes only proximity into account.
- If you specify the optional SDO_BATCH_SIZE parameter, SDO_NN keeps returning neighbors in the order of distance to the WHERE clause. When SDO_BATCH_SIZE is specified, use the ROWNUM pseudocolumn to limit the number of geometries to return. For example:

```
WHERE SDO_NN (…'SDO_BATCH_SIZE=10') = 'TRUE'
and bank_name = 'CHASE' and rownum < 6
```

  Using the preceding example, you get back the five closest banks with bank_name = 'CHASE'.
- SDO_BATCH_SIZE=0 enables SDO_NN to self-tune, automatically increasing the batch size as it proceeds during execution.

## SDO_NN: Optional Arguments

- UNIT (optional with SDO_NN_DISTANCE):
  - Is applied to SDO_NN_DISTANCE
  - Is the unit of measurement to associate with distance
- TAG (required with SDO_NN_DISTANCE):
  - Must match the integer tag specified in SDO_NN_DISTANCE
  - Associates an ancillary operator with a specific instance of SDO_NN

ORACLE



### SDO_NN: Optional Arguments (continued)

The following SDO_NN parameters are associated with the SDO_NN_DISTANCE ancillary operator, which is discussed in the next slide.

**UNIT (optional):** Unit of measurement for the results returned by the SDO_NN_DISTANCE ancillary operator. SDO_NN_DISTANCE is discussed in an upcoming slide. The default unit is the unit of measurement associated with the data. For longitude or latitude data, the default unit is meters.

**TAG (required if SDO_NN_DISTANCE is specified):** Integer value that associates SDO_NN with a particular instance of SDO_NN_DISTANCE. The tag value specified in SDO_NN must match the tag value specified in SDO_NN_DISTANCE.

# Nearest-Neighbor Ancillary Operator: SDO_NN_DISTANCE

```
SDO_NN_DISTANCE (<TAG>) returns NUMBER
```

- This returns the distance associated with the nearest neighbors returned by SDO_NN.
- This is valid only within a call to the SDO_NN operator.
- Distance conforms to the UNIT parameter in the SDO_NN operator.
- If UNIT is not specified, SDO_NN_DISTANCE uses:
  - Meters for geodetic data
  - The coordinate system unit for projected data
- TAG is a number that associates the ancillary operator with a call to SDO_NN.

ORACLE



## Nearest-Neighbor Ancillary Operator: SDO_NN_DISTANCE

SDO_NN_DISTANCE is an ancillary operator to the SDO_NN operator. This operator can be used only within a call to the SDO_NN operator. The SDO_NN_DISTANCE operator returns the distance associated with the nearest neighbors returned by SDO_NN. You can use a bind variable to store and operate on the returned distance value. The distance unit conforms to the UNIT parameter of SDO_NN. If UNIT was not specified in SDO_NN, SDO_NN_DISTANCE defaults to the following units:

- Meters for geodetic data
- The coordinate system unit for projected data

**TAG:** This parameter is of the NUMBER type and has no special meaning. You can choose any arbitrary number for the NUMBER parameter. The only requirement is that it matches the last parameter in the call to the SDO_NN operator (associates SDO_NN_DISTANCE with the SDO_NN operator).

## SDO_NN: Example

- Find the five cities nearest to Interstate I170:

```
SELECT /*+ ordered */
       c.city, c.state_abrv
FROM geod_interstates i,
     geod_cities c
WHERE i.highway = 'I170'
  AND sdo_nn(c.location, i.geom,
             'sdo_num_res=5') = 'TRUE';
```

- For optimal performance, make sure that you have a nonspatial index on `GEOD_INTERSTATES(HIGHWAY)`.

ORACLE



### SDO_NN: Example

This query finds the five cities nearest to Interstate I170. Because the `/*+ ordered */` hint is used, it is important to have an index on the `GEOD_INTERSTATES(HIGHWAY)` column because the hint makes the `HIGHWAY` column the query driver. This forces the query to locate highway `'I170'` before it tries to find its nearest neighbors.

In the `SDO_NN` operator, the following are the arguments:

- The search column is `c.location` from the `geod_cities` table.
- The window is `i.geom`, which is the highway `'I170'`.
- `'sdo_num_res=5'` is the number of the nearest neighbors to find.

## Using the SDO_NN_DISTANCE Ancillary Operator with SDO_NUM_RES

Find the five cities nearest to Interstate I170, ordered by distance:

```
SELECT /*+ ordered */
       c.city, c.state_abrv,
       sdo_nn_distance (1) distance_in_miles
FROM geod_interstates i,
     geod_cities c
WHERE i.highway = 'I170'
  AND sdo_nn(c.location, i.geom,
             'sdo_num_res=5 unit=mile', 1) = 'TRUE'
ORDER by distance_in_miles;
```

ORACLE



### Using the SDO_NN_DISTANCE Ancillary Operator with SDO_NUM_RES

This query finds the five cities nearest to Interstate I170, and returns the distance, ordered by distance, in miles. As mentioned previously, it is important to have an index on the GEOD_INTERSTATES(HIGHWAY) column when using the /*+ ordered */ hint. This forces the query to locate highway I170 before it tries to find the nearest neighbors.

Note that the SDO_NN_DISTANCE operator is called in the SELECT list with the TAG number 1. This returns the distance of each nearest neighbor to the window geometry.

In the SDO_NN operator, the following are the arguments:

- The search column is C.LOCATION from the GEOD_CITIES table.
- The query window is I.GEOM, which is highway 'I170'.
- 'sdo_num_res=5' defines the number of the nearest neighbors to find.
- 'unit=mile' associates a unit with distances returned by SDO_NN_DISTANCE.
- The TAG number 1 associates SDO_NN with the SDO_NN_DISTANCE operator.

Finally, the query performs an ORDER BY operation to order the results by distance.

**Note:** The city of Springfield is returned twice, each with a different distance. The GEOD_CITIES table has more than one city named Springfield, and each resides in a different state.

## Using the SDO_NN_DISTANCE Ancillary Operator with SDO_NUM_RES: Results

The query in the previous slide returns the five cities nearest to Interstate I170, ordered by distance:

| CITY | ST | DISTANCE_IN_MILES |
|---|---|---|
| St Louis | MO | 5.36297295 |
| Springfield | IL | 78.7997464 |
| Peoria | IL | 141.478022 |
| Evansville | IN | 158.22422 |
| Springfield | MO | 188.508631 |

ORACLE



### Using SDO_NN_DISTANCE Ancillary Operator with SDO_NUM_RES: Results

The slide shows the output of the query in the previous slide.

## Using SDO_NN with SDO_BATCH_SIZE

- Find the five cities nearest to Interstate I170 that have a population greater than 300,000, ordered by distance:

```
SELECT /*+ ordered */
       c.city, c.state_abrv, pop90,
       sdo_nn_distance (1) distance_in_miles
FROM geod_interstates i,
     geod_cities c
WHERE i.highway = 'I170'
  AND sdo_nn(c.location, i.geom,
             'sdo_batch_size=10 unit=mile', 1) = 'TRUE'
  AND c.pop90 > 300000
  AND rownum < 6
ORDER BY distance_in_miles;
```

- If you set SDO_BATCH_SIZE=0, SDO_NN chooses an optimal SDO_BATCH_SIZE.



### Using SDO_NN with SDO_BATCH_SIZE

This query finds the five cities nearest to Interstate I170 that have a population greater than 300,000 people, and returns the city, the population in 1990, and the distance (ordered by distance) in miles. In the SDO_NN operator, the following are the arguments:

- The search column is c.location from the geod_cities table.
- The query window is i.geom, which is highway 'I170'.
- 'sdo_batch_size=10' is the number of the nearest-neighbor distances to calculate at a time. Because the five closest cities may not have a population greater than 300,000, this parameter prompts SDO_NN to calculate 10 distances at a time until all the other conditions in the WHERE clause are met (that is, five cities are returned whose population is greater than 300,000.)
- 'unit=mile' associates mile as the unit of measurement with distances returned by SDO_NN_DISTANCE.
- The TAG number 1 associates SDO_NN to the SDO_NN_DISTANCE operator.
- AND pop90 > 300000 is the additional condition requiring the query to select only those cities for which the population is greater than 300,000.
- AND rownum < 6 is a stopping condition for the SQL statement. This enables the query to stop when there are five candidates that meet all the conditions in the WHERE clause.

## Using SDO_NN with SDO_BATCH_SIZE: Results

The query in the previous slide returns the five cities nearest to Interstate I170 that have a population greater than 300,000, ordered by distance:

| CITY | ST | POP90 | DISTANCE_IN_MILES |
|---|---|---|---|
| St Louis | MO | 396685 | 5.36297295 |
| Kansas City | MO | 435146 | 227.404883 |
| Indianapolis | IN | 741952 | 234.708666 |
| Memphis | TN | 610337 | 244.202072 |
| Chicago | IL | 2783726 | 253.547961 |

ORACLE



### Using SDO_NN with SDO_BATCH_SIZE: Results

The slide shows the output of the query in the previous slide.

## Important Points from the Previous Query

- You must have an index on `GEOD_INTERSTATES(HIGHWAY)`.
- `SDO_NN` returns rows in distance order to the `WHERE` clause.
- When using `SDO_BATCH_SIZE`:
  - As an exception, it is acceptable to have the `ROWNUM` condition followed by an `ORDER BY` operation
  - Disable all nonspatial indexes on columns that come from the same table as the `SDO_NN` search column (for example, `pop90`)

ORACLE



### Important Points from the Previous Query

You must understand the following points from the previous query:

- Because the `/*+ ordered */` hint is used, having an index on the `GEOD_INTERSTATES(HIGHWAY)` column forces the query to locate highway I170 before it tries to find the nearest neighbors.
- The `SDO_BATCH_SIZE` parameter continually returns the nearest neighbors (in distance order) to the other conditions in the `WHERE` clause.
- The query performs an `ORDER BY` operation to order the results by distance. Normally, a `rownum` predicate must not be followed by an `ORDER BY` clause in an Oracle SQL statement. The `rownum` predicate could terminate the results before the `ORDER BY` clause orders the entire result set. The `SDO_NN` spatial operator is an exception when ordering by distance because it is guaranteed to return the results in distance order in the `WHERE` clause.
- To ensure correct results when `SDO_BATCH_SIZE` is specified, no nonspatial indexes must be used on any of the columns of the table that contains the search column. You must ensure that there is no index on `pop90`, or the index must be disabled with a `no_index Optimizer` hint.

## Using the no_index Optimizer Hint

To disable a nonspatial index, use the no_index Optimizer hint.

```
SELECT /*+ ordered no_index(c pop90_idx) */
       c.city, pop90,
       sdo_nn_distance (1) distance_in_miles
FROM geod_interstates i,
     geod_cities c
WHERE i.highway = 'I170'
  AND sdo_nn(c.location, i.geom,
             'sdo_batch_size=0 unit=mile', 1) = 'TRUE'
  AND c.pop90 > 300000
  AND rownum < 6
ORDER BY distance_in_miles;
```

ORACLE



### Using the no_index Optimizer Hint

If the pop90 column from the geod_cities table is already indexed, it is important to specify the no_index hint to Oracle Optimizer to ensure that the correct rows are returned. Otherwise, the row order is not preserved from SDO_NN that uses sdo_batch_size.

When you specify the no_index Optimizer hint, you must use the table alias instead of the table name. Optimizer chooses a different execution plan.

This is the requirement that all nonspatial indexes must be disabled on the table that contains the SDO_NN search column (geometry-1).

**Note**

- Setting the batch size to 0 allows Oracle Spatial to self-tune the batch size during execution.
- When you are using an Optimizer hint, you must use the table alias name and not the table name. If you use the table name, Oracle Optimizer ignores the hint. In the slide example, note the "c" table alias in /*+ ordered no_index(c pop90_idx) */.

# Using SDO_NN with DISTANCE

Find the five cities nearest to Interstate I170 that have a population greater than 300,000 and a maximum distance of 240 miles from Interstate I170:

```
SELECT /*+ ordered  no_index(c pop90_idx) */
       c.city, pop90,
       sdo_nn_distance (1) distance_in_miles
FROM geod_interstates i,
     geod_cities c
WHERE i.highway = 'I170'
  AND sdo_nn(c.location, i.geom,
             'sdo_batch_size=0 distance=240 unit=mile',
              1) =  'TRUE'
  AND c.pop90 > 300000
  AND rownum < 6
ORDER BY distance_in_miles;
```

ORACLE



## Using SDO_NN with DISTANCE

In the slide example, the DISTANCE parameter is used. Now, it returns all the cities with a population greater that 300,000 but not farther than 240 miles from Interstate I170.

## Using SDO_NN with DISTANCE: Results

Find the five cities nearest to Interstate I170 that have a population greater than 300,000 and a maximum distance of 240 miles from Interstate I170:

| CITY | ST | POP90 | DISTANCE_IN_MILES |
|---|---|---|---|
| St Louis | MO | 396685 | 5.36297295 |
| Kansas City | MO | 435146 | 227.404883 |
| Indianapolis | IN | 741952 | 234.708666 |

ORACLE



### Using SDO_NN with DISTANCE: Results

Only three cities satisfied the two conditions. So, the query cannot retrieve the five nearest cities to Interstate I170.

# Lesson Agenda

- Spatial queries and operators
- SDO_WITHIN_DISTANCE operator
- SDO_NN operator
  - SDO_NN_DISTANCE ancillary operator
- Spatial join by using the SDO_JOIN operator
- Available features in Oracle Locator

ORACLE

## Spatial Join: SDO_JOIN Operator

- This is used to compare all geometries in one layer with all geometries in another layer.
  - This is also useful when comparing most of one layer with all or most of another layer.
- Spatial joins can be used to answer questions such as: *Which highways cross national parks?*
- Both geometry layers being compared:
  - Must be in the same coordinate system
  - Have the same dimensionality
- Both tables must be spatially indexed.

ORACLE



### Spatial Join: SDO_JOIN Operator

Spatial join is used to join all geometries in one layer with all geometries in another layer. This is unlike a query window, which compares a single geometry to all geometries of a layer. Both the geometry layers being compared must have the same SRID value and the same number of dimensions.

The SDO_JOIN operator is used to perform spatial joins. The operator requires the layers passed in to be indexed. The indexes must be R-tree indexes of the same dimensionality with the same spatial reference system ID (SRID).

SDO_JOIN takes all rows in one layer and compares them with all rows in the other layer. The driver table is determined automatically by Oracle Spatial. SDO_JOIN uses the R-tree height to determine the relative size of the two tables involved in the query and uses the smaller table as the driver table. Additionally, if the heights are the same, and a table with LAYER_GTYPE=POLYGON or LAYER_GTYPE=MULTIPOLYGON exists, it is used as the driver table.

# Syntax of the SDO_JOIN Operator

```
SDO_JOIN( <table_name-1>, <column_name-1>,
          <table_name-2>, <column_name-2>
         [,'parameters'])
RETURN SDO_ROWIDSET
```

- <table_name-n>: Tables that contain the SDO_GEOMETRY columns
- <column_name-n>: Indexed SDO_GEOMETRY columns

ORACLE



## Syntax of the SDO_JOIN Operator

The SDO_JOIN operator accepts two table names and spatial column names to perform spatial joins.

- **<table_name-n>:** Specifies each of the tables, which contain the SDO_GEOMETRY column
- **<column_name-n>:** Specifies the indexed SDO_GEOMETRY column associated with table_name-n

# Syntax of the SDO_JOIN Operator

```
SDO_JOIN( <table_name-1>, <column_name-1>,
          <table_name-2>, <column_name-2>
         [,'parameters'])
RETURN SDO_ROWIDSET
```

- 'parameters': Choose one of two:
  - MASK: This defines the topological relationship to look for—for example, ANYINTERACT, INSIDE, OVERLAPS, and so on.
  - DISTANCE and the optional UNIT: Geometry pairs are returned if they are within this distance of each other.

Note: If parameters are not specified, this is analogous to SDO_FILTER.



## Syntax of the SDO_JOIN Operator (continued)

parameters: Optional parameters are quoted strings of keywords and their values. The parameters are:

- MASK=<mask>: This defines the topological relationship of column_name-1 to column_name-2 that you want to compare for. Valid mask specifications include the following: TOUCH, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, EQUAL, INSIDE, COVEREDBY, CONTAINS, COVERS, ANYINTERACT, ON, and FILTER. By default, if the mask is set to FILTER—that is, the SDO_JOIN operator matches all geometries in one table with all geometries in the other table based on comparing only geometry approximations stored in the index (filter operation)—a primary filter operation is performed. Multiple masks can be combined with an OR logical operator such as 'mask=inside+touch'. For any other mask, it performs both primary and secondary filter operations to get accurate results.
- **DISTANCE** with optional **UNIT**: Geometry pairs are returned if they are within this distance of each other in the specified unit.
- If parameters are not specified, this is analogous to SDO_FILTER.

# Syntax of the SDO_JOIN Operator

- This returns an object of the SDO_ROWIDSET type—that is, a pair of ROWIDs that matched, from the two spatial layers.

```
DESCRIBE SDO_ROWIDSET
```

- ROWID1 refers to ROWID from table_name-1 and ROWID2 refers to ROWID from table_name-2.

```
SDO_ROWIDSET TABLE OF MDSYS.SDO_ROWIDPAIR
 Name            Null?    Type
 --------------- -------- -----------
 ROWID1                   VARCHAR2(24)
 ROWID2                   VARCHAR2(24)
```

ORACLE



## Syntax of the SDO_JOIN Operator (continued)

SDO_JOIN returns an object of SDO_ROWIDSET, which consists of a table of objects of SDO_ROWIDPAIR. SDO_ROWIDPAIR is essentially a pair of ROWIDs that matched from the two spatial layers. It consists of two fields, ROWID1 and ROWID2, of the VARCHAR2 data type. Therefore, the SDO_JOIN operator returns the ROWID pairs that match the join condition between the two tables.

**Note:** A ROWID is an implicit primary key of a table.

# Spatial Join: Using the MASK Parameter

- Find all the city-and-county pairs that have any interaction:

```
SELECT /*+ ordered use_nl(a,b) use_nl(a,c)*/
       b.city, c.county
FROM TABLE(SDO_JOIN(
          'GEOD_COUNTIES', 'GEOM',
          'GEOD_CITIES', 'LOCATION',
          'MASK=ANYINTERACT')) a,
     geod_cities b,
     geod_counties c
WHERE a.rowid1 = c.rowid
  AND a.rowid2 = b.rowid
ORDER BY b.city;
```

- Use the TABLE syntax to flatten SDO_ROWIDSET into ROWID1-and-ROWID2 pairs.

ORACLE



## Spatial Join: Using the MASK Parameter

In this example, all the city-and-county pairs are returned where cities and counties have any interaction. When writing SDO_JOIN queries, remember the following points:

- For best performance, use the ORDERED and USE_NL optimizer hints.
- Specify the SDO_JOIN operator first in the FROM clause.
- Use the TABLE syntax to flatten the SDO_ROWIDSET returned by SDO_JOIN into ROWID1-and-ROWID2 pairs.

**Note:** Better performance has been observed with the ORDERED and USE_NL optimizer hints.

# Spatial Join: Using the DISTANCE and UNIT Parameters

Find all cities within 10 miles of all interstate highways:

```
SELECT /*+ ordered use_nl(a,b) use_nl(a,c)*/
       b.city, c.highway
FROM TABLE(SDO_JOIN(
             'GEOD_INTERSTATES', 'GEOM',
             'GEOD_CITIES', 'LOCATION',
             'DISTANCE=10 UNIT=MILE')) a,
     geod_cities b,
     geod_interstates c
WHERE a.rowid1 = c.rowid
  AND a.rowid2 = b.rowid
ORDER BY b.city;
```

ORACLE



## Spatial Join: Using the DISTANCE and UNIT Parameters

In this example, all the city-highway pairs are returned, where cities are within 10 miles of a highway. Because no mask is specified, by default, the mask is FILTER. In this case, only the primary filter operation is performed. Therefore, the query returns approximate results.

# Important Points About Oracle Spatial Operators

There are three important points to remember that help to ensure optimal performance when using the Oracle Spatial operators:

- Always use `= 'TRUE'`, and never `<> 'FALSE'` or `= 'true'`.
- Use the `/*+ ORDERED */` hint when the query window comes from a table.
- When using `SDO_NN` with the `SDO_BATCH_SIZE` parameter, disable all nonspatial indexes on columns that come from the same table as the `SDO_NN` search column.
  - This can be done with a `no_index` Optimizer hint.

ORACLE



### Important Points About Oracle Spatial Operators

There are three points to keep in mind when writing Oracle Spatial queries by using the previous operators:

- Always use = `'TRUE'` when testing spatial operators. Never use `<>'FALSE'` or `='true'`.
- Use the `/*+ ordered */` hint when the query window comes from a table. This may not always be necessary, but it never impacts performance.
- When using `SDO_NN` with the `SDO_BATCH_SIZE` parameter in a SQL statement, disable all nonspatial indexes on columns that come from the same table as the `SDO_NN` search column. This can be done with a `no_index` Optimizer hint.

# Parallelism with Spatial Operators and CREATE TABLE AS SELECT

- In general, spatial operators do not leverage parallelism.
- But you can leverage parallelism with spatial operators with CREATE TABLE AS SELECT.
- Parallelism with spatial operators and CREATE TABLE AS SELECT occurs when multiple query windows feed the second argument of the spatial operator—for example:

```
ALTER SESSION ENABLE PARALLEL QUERY;

CREATE TABLE results NOLOGGING PARALLEL 4 AS
SELECT /*+ ordered */ a.state, b.county
FROM geod_states a,
     geod_counties b
WHERE sdo_relate (b.geom, a.geom, 'mask=touch')='TRUE';
```

ORACLE



## Parallelism with Spatial Operators and CREATE TABLE AS SELECT

In general, spatial operators do not leverage parallelism. But you can leverage parallelism with spatial operators together with CREATE TABLE AS SELECT.

Parallelism with spatial operators and CREATE TABLE AS SELECT occurs when multiple query windows feed the second argument of the spatial operator. The recommendation is to set a parallel degree to the number of free CPUs that you have access to.

CREATE TABLE AS SELECT divides the windows passed into the second argument of the spatial operator across the available CPUs. For example, in this slide, 50 states feed the second argument of SDO_RELATE. 50 searches are performed, one for each state. The 50 searches are divided across the available CPUs, but each CPU processes only one state at a time. So the query takes at least as long as the time it takes to complete the slowest of the 50 searches. But this is still faster than serializing all 50 searches on a single CPU.

# Lesson Agenda

- Spatial queries and operators
- SDO_WITHIN_DISTANCE operator
- SDO_NN operator
  - SDO_NN_DISTANCE ancillary operator
- Spatial join by using the SDO_JOIN operator
- Available features in Oracle Locator

# Available Features in Oracle Locator

Everything covered in the course till the end of this lesson is included in Oracle Locator (available at no extra cost):

- Includes all spatial operators:
  - SDO_FILTER
  - SDO_RELATE and the SDO_<RELATIONSHIP> operators
  - SDO_WITHIN_DISTANCE
  - SDO_NN and SDO_NN_DISTANCE
  - SDO_JOIN
- Includes geometry validation routines
- Provides the functionality required to support all major GIS vendors' solutions

  **Note:** In the forthcoming lessons, you learn about the features or functionality that are part of Oracle Spatial, a licensed database option.
  - Exceptions in Oracle Locator are also listed.

ORACLE



## Available Features in Oracle Locator

Oracle Locator includes all the following spatial operators:

- SDO_FILTER
- SDO_RELATE and SDO_RELATIONSHIP operators
- SDO_WITHIN_DISTANCE
- SDO_NN and SDO_NN_DISTANCE
- SDO_JOIN

Oracle Locator includes all validation routines.

Oracle Locator provides all functionality required to support all major GIS vendors' solutions.

**Note:** For more information about what features are officially available in Oracle Locator, refer to the Oracle Locator appendix in the *Oracle Spatial Developer's Guide.*

# Summary

In this lesson, you should have learned to use:

- `SDO_WITHIN_DISTANCE` to solve distance-related queries
- The `SDO_NN` operator to perform the nearest-neighbor query
- The `SDO_JOIN` operator to get a spatial cross-product

ORACLE



## Summary

In this lesson, you learned how to use `SDO_WITHIN_DISTANCE`, `SDO_NN`, and `SDO_JOIN` to solve spatial queries.

# Practice 10: Overview

This practice covers the following topics:

- Running the `SDO_NN` and `SDO_WITHIN_DISTANCE` queries
- Running the `SDO_JOIN` queries

ORACLE



## Practice 10: Overview

In this practice, you run queries that involve the `SDO_NN`, `SDO_WITHIN_DISTANCE`, and `SDO_JOIN` operators.

## Practice 10

1. Find cities within 100 miles of the highway `'I10/I5'`. Use the `geod_interstates` and `geod_cities` tables.
2. Find the five nearest cities and their distances to the highway `'I170'`. Order by distance in miles.
3. Find the five cities nearest to the highway `'I170'` whose population is greater than 300,000 and return their distance to highway `'I170'`.
4. Find all city-and-county pairs that have any interaction.
5. Find all cities within 20 miles of all interstate highways.