
Oracle9i: Spatial

Student Guide
Volume 1

D12848GC10
Edition 1.0
November 2001
D34080

ORACLE®

Authors

Dan Geringer

**Technical Contributors
and Reviewers**

Daniel Abugov
Nicole Alexander
Daniel Geringer

Publisher

Kekoa Lavatai

Copyright © Oracle Corporation, 2001. All rights reserved.

This documentation contains proprietary information of Oracle Corporation. It is provided under a license agreement containing restrictions on use and disclosure and is also protected by copyright law. Reverse engineering of the software is prohibited. If this documentation is delivered to a U.S. Government Agency of the Department of Defense, then it is delivered with Restricted Rights and the following legend is applicable:

Restricted Rights Legend

Use, duplication or disclosure by the Government is subject to restrictions for commercial computer software and shall be deemed to be Restricted Rights software under Federal law, as set forth in subparagraph (c)(1)(ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

This material or any portion of it may not be copied in any form or by any means without the express prior written permission of Oracle Corporation. Any other copying is a violation of copyright law and may result in civil and/or criminal penalties.

If this documentation is delivered to a U.S. Government Agency not within the Department of Defense, then it is delivered with "Restricted Rights," as defined in FAR 52.227-14, Rights in Data-General, including Alternate III (June 1987).

The information in this document is subject to change without notice. If you find any problems in the documentation, please report them in writing to Education Products, Oracle Corporation, 500 Oracle Parkway, Box SB-6, Redwood Shores, CA 94065. Oracle Corporation does not warrant that this document is error-free.

Oracle and all references to Oracle Products are trademarks or registered trademarks of Oracle Corporation.

All other products or company names are used for identification purposes only, and may be trademarks of their respective owners.

Contents

Preface

1 Introduction to Oracle Spatial

- Course Agenda 1-2
- Course Objectives 1-4
- History and Positioning 1-5
- Relational Model 1-7
- Additional Resources 1-8

2 Oracle Spatial Concepts

- Objectives 2-2
- Types of Location Data 2-3
- Geometric Primitive Types 2-5
- Spatial Data Model 2-11
- Elements 2-12
- Geometry 2-13
- Layers 2-14
- Optimized Query Model 2-16
- Spatial Indexing 2-18
- Quadtree Decomposition of Space 2-19
- R-tree: Generates Minimum Bounding Rectangle (MBR) Tile 2-20
- Primary and Secondary Filter Concept Quadtree Example 2-21
- Spatial Window Queries 2-23
- Spatial Joins (Spatial Cross Product) 2-24
- Linear Referencing System (LRS) 2-25
- Summary 2-26

3 Creating Spatial Layers

- Objectives 3-2
- SDO_GEOMETRY Object 3-3
- SDO_GTYPE 3-7
- Linear Referencing System (LRS) 3-9
- SDO_GTYPE with LRS: Oracle9i 3-10
- LRS Measure Positions: Summary 3-11
- New Methods on SDO_GTYPE 3-12
- SDO_GEOMETRY Object 3-14
- Element Example: Point 3-17
- Element Types Summarized 3-19
- Element Example: Line String 3-21
- Element Example: Arc String 3-22
- Element Example: Polygon 3-23
- Element Example: Arc Polygon 3-24
- Element Example: Rectangle 3-25
- Element Example: Circle 3-26
- Element Example: Compound Line String 3-27
- Element Example: Compound Polygon 3-28

- Details of Polygon Element Types 3-30
- Element Example: Polygon with a Void 3-31
- Element Example: Compound Polygon with a Void 3-32
- Element Example: Unknown Element 3-33
- Collection Example: Multipoint 3-34
- Collection Example: Multiline String 3-35
- Collection Example: Multipolygon 3-36
- Constructing Geometries 3-37
- Element Types Summarized 3-39
- Syntax for Flattening the VARRAYs 3-41
- Spatial Metadata 3-42
- USER_SDO_GEOM_METADATA 3-43
- USER_SDO_GEOM_METADATA SDO_DIM_ELEMENT Object 3-44
- Populating the USER_SDO_GEOM_METADATA View 3-46
- Notes on Dimensions 3-47
- *_SDO_GEOM_METADATA System Views 3-49
- Summary 3-50

4 Coordinate Systems Overview

- Objectives 4-2
- Coordinate Systems Concepts 4-3
- Projected CS Concepts 4-5
- Geodetic CS Concepts 4-6
- Geodetic Geometry Examples 4-9
- Coordinate Systems in Oracle Spatial 4-11
- MDSYS.CS_SRS Table 4-12
- Associating Geometries with Coordinate Systems 4-13
- Adding Coordinate Systems Information to Existing Layers 4-14
- Adding Coordinate Systems Information Example 4-15
- Common Coordinate Systems 4-17
- Restrictions on Geodetic Coordinate Systems 4-18
- Summary 4-19

5 Loading Spatial Data

- Objectives 5-2
- Loading Spatial Data 5-3
- SQL*Loader 5-4
- SQL*Loader Features 5-5
- SQL*Loader Limitation 5-7
- SQL*Loader for Point Data 5-8
- SQL*Loader for Lines and Polygons (Oracle9i) 5-10
- SQL*Loader for Lines and Polygons (Oracle 8i) 5-11
- SQL*Loader Restrictions 5-12
- Export/Import 5-13
- Transactional Inserts 5-15
- Shapefile Converter 5-18
- Invoking the Converter 5-19

- Example Run 5-22
- Log Output 5-23
- Usage Notes 5-24
- Interactive Run 5-26
- Validating Geometries 5-27
- The VALIDATE_GEOMETRY Function 5-28
- VALIDATE_GEOMETRY Example 5-29
- The VALIDATE_LAYER Procedure 5-30
- The Results Table 5-31
- VALIDATE_LAYER Example 5-32
- VALIDATE_GEOMETRY Example Similar to VALIDATE_LAYER 5-33
- Spatial Migration Utility (SDO_MIGRATE.TO_CURRENT) 5-34
- Spatial Migration Utility (Relational Schema to Current) 5-35
- Migration Example (Relational Schema To Current) 5-39
- Data Migration 8.1.5 SDO_GEOMETRY to Current 5-40
- Manual Data Migration 8.1.5 to 8.1.6 and Higher 5-41
- Summary 5-42

6 Indexing Spatial Data

- Objectives 6-2
- Spatial Indexing 6-3
- Quadtree Indexing 6-4
- Tessellation 6-5
- How a Geometry Is Indexed Using Quadtree Indexing 6-6
- Primary Filter Example (Quadtree Index) 6-7
- A Look at Quadtree Index Structures 6-8
- What Happens During Quadtree 6-10
- How Much of my Quadtree Index Has Been Built? 6-11
- R-tree Indexing 6-12
- R-tree Indexing Concept 6-13
- How Geometries Are Indexed Using R-trees 6-14
- Primary Filter Example (R-tree Index) 6-16
- Quadtrees or R-trees? 6-17
- CREATE INDEX 6-22
- CREATE INDEX (Quadtree) 6-24
- CREATE INDEX (Quadtree: Hybrid Indexes Only) 6-26
- CREATE INDEX (R-tree) 6-28
- CREATE INDEX 6-29
- DROP INDEX 6-32
- ALTER INDEX 6-33
- Spatial Index Dictionary Views 6-37
- USER_SDO_INDEX_METADATA (Spatial Index Dictionary View) 6-38
- Spatial Index Informational Views 6-39
- Summary 6-40

7 Tuning and Administration

- Objectives 7-2
- Tuning and Administration Tools 7-3
- What Type of Index Should I Use? 7-4
- Tuning Assistants The SDO_TUNE Package 7-5
- EXTENT_OF 7-6
- EXTENT_OF Example 7-7
- Flexible Fixed Indexing (Quadtree) 7-8
- ESTIMATE_TILING_LEVEL (Quadtree Only) 7-9
- Type of Estimation 7-11
- ESTIMATE_TILING_LEVEL Example 7-12
- Primary and Secondary Filter Concept Quadtree Example 7-13
- Oracle Enterprise Manager (OEM) Spatial Index Advisor 7-15
- Summary 7-24

8 Spatial Queries

- Objectives 8-2
- Data Sets Loaded So Far 8-3
- Optimized Query Model 8-4
- Primary Filter Example (Quadtree Index) 8-6
- Primary Filter Example (R-tree Index) 8-7
- Spatial Operators Versus Functions 8-8
- The SDO_FILTER Operator 8-11
- Required Arguments 8-12
- Optional Arguments 8-14
- SDO_FILTER Example 8-16
- SDO_CS.VIEWPORT_TRANSFORM 8-17
- SDO_CS.VIEWPORT_TRANSFORM Function 8-18
- SDO_CS.VIEWPORT_TRANSFORM <to_srid> Is Geodetic 8-19
- Explicit Viewport Transformation (Query Against a Geodetic Layer) 8-20
- SDO_CS.VIEWPORT_TRANSFORM <to_srid> Is Projected 8-21
- Explicit Viewport Transformation (Query Against a Projected Layer) 8-22
- Spatial (Topological) Relationships 8-23
- The SDO_RELATE Operator 8-26
- Required Arguments 8-27
- Optional Arguments 8-31
- SDO_RELATE: A Window Query 8-33
- SDO_RELATE and PL/SQL 8-37
- SDO_RELATE: Window Query 8-38
- SDO_RELATE: Join Query 8-40
- The SDO_WITHIN_DISTANCE Operator 8-41
- Arguments 8-42
- Optional Arguments 8-43
- SDO_WITHIN_DISTANCE Examples 8-44
- The SDO_NN (Nearest Neighbor) Operator 8-45

- Arguments 8-46
- SDO_NN_DISTANCE Nearest Neighbor Ancillary Operator 8-49
- SDO_NN Example 8-50
- SDO_NN Example: Results 8-51
- SDO_NN Example 8-52
- SDO_NN Example: Results 8-54
- The SDO_GEOM.RELATE Function 8-55
- SDO_GEOM.RELATE Parameters 8-56
- SDO_GEOM.RELATE Function 8-57
- Oracle 9i Intermedia Locator 8-60
- Summary 8-61

9 Spatial Analysis

- Objectives 9-2
- Area and Length Calculations 9-3
- The SDO_AREA Function 9-5
- SDO_AREA Examples 9-6
- The SDO_LENGTH Function 9-8
- SDO_LENGTH Examples 9-9
- Distance Calculations 9-10
- The SDO_DISTANCE Function 9-11
- SDO_DISTANCE Examples 9-12
- Accuracy of Geodetic Area Calculations 9-14
- Accuracy of Geodetic Length and Distance Calculations 9-15
- Arc Densification and Buffering 9-16
- Arc Densification 9-17
- The SDO_ARC_DENSIFY Function 9-19
- SDO_ARC_DENSIFY 9-21
- SDO_ARC_DENSIFY: Example 9-22
- SDO_BUFFER Function 9-23
- Buffer Examples 9-26
- SDO_BUFFER Examples 9-28
- Recap of Four Important Hints 9-32
- Spatial Boolean Functions 9-33
- The SDO_<BOOLEAN> Functions 9-38
- Putting it all Together 9-39
- Spatial Analysis Functions 9-40
- Spatial MBR Functions 9-42

- Spatial MBR Function Examples 9-44
- Spatial Aggregate Functions 9-45
- SDOAGGRTYPE 9-46
- SDO_AGGR_UNION 9-47
- SDO_AGGR_UNION Example 9-48
- SDO_AGGR_CENTROID 9-50
- SDO_AGGR_CENTROID Example 9-51
- SDO_AGGR_CONVEXHULL 9-53
- SDO_AGGR_CONVEXHULL Example 9-54
- SDO_AGGR_MBR 9-55
- SDO_AGGR_MBR Example 9-56
- Summary 9-57

10 Advanced Coordinate Systems Concepts

- Objectives 10-2
- Whole Earth Geometry Model 10-3
- Tolerance Revisited: Projected Versus Geodetic 10-5
- Migration from 8.1.x: Some Considerations 10-6
- Coordinate System Transformations 10-7
- Coordinate System Transformations Functions 10-8
- The SDO_CS.TRANSFORM Function 10-9
- SDO_CS.TRANSFORM Example 10-10
- The SDO_CS.TRANSFORM_LAYER Procedure 10-11
- TRANSFORM_LAYER Example 10-12
- Transformations 10-13
- Transformation Example 10-14
- The SDO_CS.VIEWPORT_TRANSFORM Function 10-15
- SDO_CS.VIEWPORT_TRANSFORM When <to_srid> Is Geodetic 10-17
- SDO_CS.VIEWPORT_TRANSFORM When <to_srid> Is Projected 10-18
- Explicit Viewport Transformation 10-19
- Indexing Geodetic Data 10-20
- UNIT Support in Oracle Spatial 10-23
- Accuracy of Geodetic Area Calculations 10-26
- Accuracy of Geodetic Length and Distance Calculations 10-27
- Geodetic or Projected Coordinate Systems? 10-28
- User-Defined Coordinate Systems 10-29
- Ellipsoids 10-31
- Datums 10-32
- Projections 10-35
- Well-Known Text (WKT) for a Coordinate System 10-37
- Well-Known Text Example 10-38
- Creating a User-Defined Coordinate System 10-40
- User-Defined Coordinate Systems Example 10-42
- Local Coordinate Systems 10-44
- Local CS Examples 10-46
- Summary 10-47

11 Advanced Spatial Indexing Concepts

| | |
|---|-------|
| Objectives | 11-2 |
| Spatial Index Partitioning | 11-3 |
| Oracle Table Partitioning and Oracle Spatial | 11-4 |
| Why Partition Spatial Indexes? | 11-5 |
| Oracle9i Spatial Index Partitioning | 11-6 |
| Data Set Description for Partitioned Spatial Index Examples | 11-7 |
| Fabricated Yellow Pages Data (Businesses in New York) | 11-8 |
| The YELLOW_PAGES Table | 11-9 |
| Creating a Partitioned Spatial Index | 11-10 |
| Spatial Index Partitioning Parameters | 11-11 |
| Oracle9i Spatial Index Partitioning | 11-12 |
| Spatial Index Partitioning Syntax | 11-14 |
| Spatial Index Partitioning Restrictions | 11-16 |
| Query Examples Using Partitioned and Nonpartitioned Tables | 11-17 |
| Nonpartitioned Example | 11-18 |
| Partitioned Example | 11-19 |
| Partitioned Example: SDO_NN | 11-20 |
| Function-Based Indexes | 11-24 |
| Function-Based Indexes Procedure | 11-27 |
| Function-Based Indexes Example | 11-28 |
| Privileges for Function-Based Indexes | 11-32 |
| Performance of Function-Based Indexes | 11-33 |
| Embedded SDO_GEOMETRY Objects | 11-34 |
| Embedded Spatial Geometry | 11-35 |
| Embedded Geometry Procedure | 11-36 |
| Embedded Geometry Example | 11-37 |
| Summary | 11-39 |

12 Linear Referencing System (LRS)

| | |
|---|-------|
| Objectives | 12-2 |
| What Is LRS? | 12-3 |
| LRS Application | 12-6 |
| LRS Segment | 12-7 |
| Associating Events with LRS | 12-8 |
| Defining an LRS Geometry | 12-9 |
| LRS Concepts | 12-10 |
| SDO_GTYPE with LRS: Oracle9i and Beyond | 12-13 |
| LRS Measure Positions: Summary | 12-15 |
| LRS Support for Collections | 12-16 |
| Defining LRS Structures | 12-18 |
| Constructing Geometries with LRS: All Measures Are Known | 12-19 |
| Constructing Geometries with LRS: Some Measures Are Unknown | 12-20 |
| SDO_LRS.CONVERT_TO_LRS_GEOM | 12-21 |
| SDO_LRS.CONVERT_TO_LRS_LAYER | 12-23 |

- Overview of LRS Functions: Geometry Manipulation 12-24
- Overview of LRS Functions: Geometry Interrogation 12-27
- LRS In Oracle Spatial: Linear Functions 12-28
- Overview of LRS Functions: Point Functions 12-29
- LRS in Oracle Spatial: Point Functions 12-31
- Overview of LRS Functions: Inspector Functions 12-32
- Overview of LRS Functions: Validation Functions 12-34
- LRS Application 12-36
- Case Study 1 12-37
- Case Study 2 12-39
- LRS Example 12-41
- Summary 12-46

13 Other Features

- Objectives 13-2
- Workspace Manager 13-3
 - What Is Workspace Manager? 13-4
 - What Is a Database Workspace? 13-5
 - How Does Workspace Manager Work 13-7
 - Workspace Manager Benefits: Applications 13-9
 - Workspace Manager Benefits: Developer and DBA 13-10
 - Workspace Manager Primitives 13-11
 - Enterprise Manager Interface 13-13
 - Architecture 13-15
 - Workspace Manager Administrator Role 13-16
 - Workspace-Enable a Table 13-17
 - Guidelines for Tables Participating in a Workspace 13-19
 - Disabling Workspace Participation for a Table 13-21
 - Create a Workspace 13-22
 - Assign Workspace: Associate a User 13-23
 - Assign Workspace: Grant Privileges 13-24
 - Assign Workspace: Set Locks 13-25
 - Create Workspace Savepoint 13-26
 - Examples of Implicit and Explicit Savepoints 13-27
 - Compare Savepoints: Find Differences 13-28
 - Delete Savepoint 13-30
 - Freeze a Workspace 13-31
 - Rollback a Workspace 13-32
 - Refresh a Workspace 13-33
 - Resolve Workspace Conflicts 13-34
 - Conflict Resolution Example: Check for Existence of Conflicts 13-35
 - Conflict Resolution Example: Resolve Conflicts 13-36
 - Merge a Workspace 13-37
 - Workspace Views 13-38

| | |
|--|-------|
| Compress Workspace or Workspace Tree | 13-39 |
| Compress Workspace Tree Example | 13-40 |
| Other Workspace Tasks | 13-41 |
| Image/Raster Data Support | 13-42 |
| Basic Image/Raster Storage and Retrieval | 13-43 |
| Geocoding | 13-44 |
| What Is Geocoding? | 13-45 |
| Summary | 13-46 |

Excercises

Preface

Profile

Before You Begin This Course

Before you begin this course, you should have the following qualifications:

- Understanding of Oracle Database Concepts
- Understanding of SQL

Prerequisites

- Mandatory courses:
 - Introduction to Oracle9i: SQL
- Suggested courses:
 - Oracle9i SQL Tuning
 - Oracle9i DBA Fundamentals I

How This Course Is Organized

Oracle9i Spatial is an instructor-led course featuring lecture and hands-on exercises. Online demonstrations and written practice sessions reinforce the concepts and skills introduced.

Related Publications

Oracle Publications

Title

Oracle Spatial User's Guide and Reference

Part Number

A88805-01

Typographic Conventions

Typographic Conventions in Text

| Convention | Element | Example |
|--|---|---|
| Bold italic | Glossary term (if there is a glossary) | The <i>algorithm</i> inserts the new key. |
| Caps and lowercase | Buttons, check boxes, triggers, windows | Click the Executable button. Select the Can't Delete Card check box. Assign a When-Validate-Item trigger to the ORD block. Open the Master Schedule window. |
| Courier new, case sensitive (default is lowercase) | Code output, directory names, filenames, passwords, pathnames, URLs, user input, usernames | Code output: <code>debug.set ('I', 300);</code> Directory: <code>bin</code> (DOS), <code>\$FMHOME</code> (UNIX) Filename: Locate the <code>init.ora</code> file. Password: User <code>tiger</code> as your password. Pathname: Open <code>c:\my_docs\projects</code> URL: Go to <code>http://www.oracle.com</code> User input: Enter <code>300</code> Username: Log on as <code>scott</code> |
| Initial cap | Graphics labels (unless the term is a proper noun) | Customer address (<i>but</i> Oracle Payables) |
| Italic | Emphasized words and phrases, titles of books and courses, variables | Do <i>not</i> save changes to the database. For further information, see <i>Oracle7 Server SQL Language Reference Manual</i> . Enter <code>user_id@us.oracle.com</code> , where <i>user_id</i> is the name of the user. |
| Quotation marks | Interface elements with long names that have only initial caps; lesson and chapter titles in cross-references | Select "Include a reusable module component" and click Finish. This subject is covered in Unit II, Lesson 3, "Working with Objects." |
| Uppercase | SQL column names, commands, functions, schemas, table names | Use the <code>SELECT</code> command to view information stored in the <code>LAST_NAME</code> column of the <code>EMP</code> table. |

| Convention | Element | Example |
|------------|------------------|---|
| Arrow | Menu paths | Select File—> Save. |
| Brackets | Key names | Press [Enter]. |
| Commas | Key sequences | Press and release keys one at a time: [Alternate], [F], [D] |
| Plus signs | Key combinations | Press and hold these keys simultaneously: [Ctrl]+[Alt]+[Del] |

Typographic Conventions in Code

| Convention | Element | Example |
|--------------------|----------------------------|--|
| Caps and lowercase | Oracle Forms triggers | When-Validate-Item |
| Lowercase | Column names, table names | SELECT last_name FROM s_emp; |
| | Passwords | DROP USER scott IDENTIFIED BY tiger; |
| | PL/SQL objects | OG_ACTIVATE_LAYER (OG_GET_LAYER ('prod_pie_layer')) |
| Lowercase italic | Syntax variables | CREATE ROLE <i>role</i> |
| Uppercase | SQL commands and functions | SELECT userid FROM emp; |

Typographic Conventions in Navigation Paths

This course uses simplified navigation paths, such as the following example, to direct you through Oracle Applications.

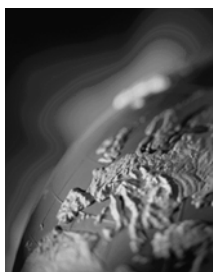
(N) Invoice—>Entry—>Invoice Batches Summary (M) Query—>Find (B) Approve

This simplified path translates to the following:

1. (N) From the Navigator window, select Invoice—>Entry—>Invoice Batches Summary.
2. (M) From the menu, select Query—>Find.
3. (B) Click the Approve button.

N = Navigator, **M** = Menu, **B** = Button

Oracle9i Spatial Training



ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

1

Introduction to Oracle Spatial

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Course Agenda

Lesson 1: Course Introduction

Lesson 2: Oracle Spatial Concepts

Lesson 3: Creating Spatial Layers

Lesson 4: Coordinate Systems Overview

Lesson 5: Loading Spatial Data

Lesson 6: Indexing Spatial Data

Lesson 7: Tuning and Administration

ORACLE

1-2

Copyright © Oracle Corporation, 2001. All rights reserved.

How This Course Is Organized

Lesson 1 is a general introduction to the course.

Lesson 2 is an high level introduction to Oracle Spatial concepts, and includes an overview of all of the material that will be covered in this course. You are not expected to understand all the concepts by the end of Lesson 2. All of the material in Lesson 2 will be discussed in detail in future lessons.

Lesson 3 discusses creating Oracle tables with a spatial column.

Lesson 4 is a brief introduction to coordinate systems.

Lesson 5 discusses loading spatial data into Oracle.

Lesson 6 discusses indexing spatial data in Oracle.

Lesson 7 discusses tuning and administration of spatial data in Oracle.

Course Agenda

Lesson 8: Spatial Queries

Lesson 9: Spatial Analysis

Lesson 10: Advanced Coordinate Systems Concepts

Lesson 11: Advanced Spatial Indexing Concepts

Lesson 12: Linear Referencing

Lesson 13: Other Features

ORACLE

1-3

Copyright © Oracle Corporation, 2001. All rights reserved.

How This Course Is Organized (continued)

Lesson 8 discusses querying spatial data in Oracle.

Lesson 9 discusses spatial analysis functions.

Lesson 10 discusses advanced coordinate system concepts.

Lesson 11 discusses advanced spatial indexing concepts.

Lesson 12 discusses Linear Referencing.

Lesson 13 discusses other features supported/used by Oracle Spatial.

Course Objectives

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

- **Identify spatial data concepts**
- **Use the object-relational spatial model to**
 - **Create Oracle tables with spatial columns**
 - **Load and index spatial data**
 - **Tune spatial databases**
 - **Query and analyze spatial data**

ORACLE

1-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Course Objectives

This course introduces you to the major components of Oracle Spatial, an innovative technology for storing and manipulating large volumes of spatial information. After completing this course, you should be able to:

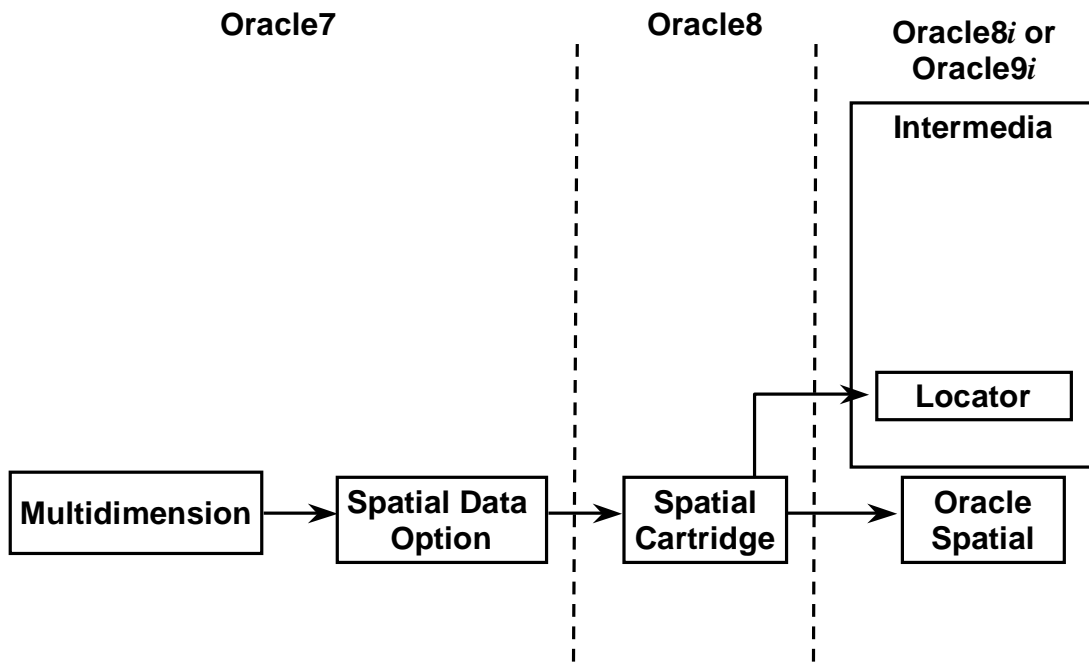
Identify the fundamental features of Oracle9i Spatial

- The SDO_GEOMETRY type
- Identify each geometric primitive supported by Oracle9i Spatial
- Create, alter, and drop Oracle tables with spatial columns.
- Load spatial data
- Index spatial data
- Tune spatial data
- Query, manipulate, and extract spatial data by using geometric operators and functions

Additionally, you should understand the following concepts associated with Oracle9i Spatial

- The spatial data model
- The spatial query model
- What a spatial index is, and when/why it is used
- How coordinate systems are used

History and Positioning



ORACLE

1-5

Copyright © Oracle Corporation, 2001. All rights reserved.

History and Positioning

Oracle Spatial is the current product name. The product name has evolved over time: Multidimension (MD), Spatial Data Option (SDO), Spatial Cartridge, and 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 lots of point data more efficiently in Oracle. Together with Oracle consulting, this project evolved into the first version of the product, called Multidimension (MD).

Oracle quickly realized that a point only solution did not satisfy most of our customer's requirements. 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).
- Oracle8i: The name was changed to Oracle Spatial and offers major new features including a new object-relational data model for Oracle Spatial. Geometries are now stored in a single row/single column.

History and Positioning (continued)

Intermedia Locator is a subset of Oracle Spatial. It can store and index all spatial data types. It includes the spatial operators SDO_RELATE, SDO_FILTER, SDO_NN, and SDO_WITHIN_DISTANCE. Any requirements beyond this requires Oracle Spatial.

Relational Model

- **Older technology**
 - Difficult to use
 - Does not support all element types
 - No support for newer features
 - No OEM support
 - In process of being deprecated
- **This course requires no knowledge of the relational model.**

ORACLE

1-7

Copyright © Oracle Corporation, 2001. All rights reserved.

The Relational Model

- The older relational data model is much more difficult to use. With Oracle's extensible indexing, the object-relational model automates much of the work that you had to do with the relational model (for example, the index is maintained, queries are much simpler, and so on).
- New element types are introduced in the object-relational model (for example, circles, circular arcs, compound linestrings, and compound polygons).
- No new features or performance enhancements are being incorporated into the relational model.
- In Oracle 8.1.6, OEM introduced tuning and schema manager support for the Oracle Spatial object-relational model only.
- The relational model is in the process of being deprecated.

Note: Understanding the relational model is *not* required to learn the Oracle Spatial object-relational model.

Additional Resources

- **Oracle Technology Network**
 - <http://otn.oracle.com/products/spatial>
 - **Discussion forum at:**
<http://otn.oracle.com/products/spatial>
then click on Discussion Forum
- **Metalink**
 - <http://metalink.oracle.com>
 - **Registration required**
 - **Patch, TAR, and Bug information**
 - **Oracle Spatial is part of the core patch set starting 9.0.1.2**
- **Oracle Consulting – For Oracle Spatial expertise contact JUSTIN.WINTER@ORACLE.COM**

ORACLE

1-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Additional Resources

- Oracle Technology Network (OTN) - (<http://otn.oracle.com>) Lots of technical information available (spatial and non spatial) as well as discussion forums.
- Metalink - (<http://metalink.oracle.com>) This is where you get Oracle Spatial patches. Some Spatial patches are hardware independent. Only Solaris 32 bit and NT platform dependent patches are available for releases before Oracle 9.0.1.2. Starting with release 9.0.1.2 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 on the patches button.

Under product family, choose "Oracle Server".

Under product name, choose "Spatial".

Select your platform and your release and Submit.

Oracle consulting has a group with considerable Oracle Spatial expertise. Contact Justin.Winter@Oracle.com for more information.

2

Oracle Spatial Concepts

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Describe the supported geometric data types**
- **Define the Oracle Spatial data model**
- **Describe a layer**
- **Describe the spatial query model**
- **Describe the concept of spatial indexing**
- **Identify spatial queries and spatial joins**
- **Describe Linear Referencing System (LRS)**

ORACLE

2-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

This section is intended to introduce you to the concepts associated with Oracle Spatial. This lesson is an overview only. You are not expected to fully understand all the concepts discussed in this lesson. All concepts are discussed in much more detail in later lessons.

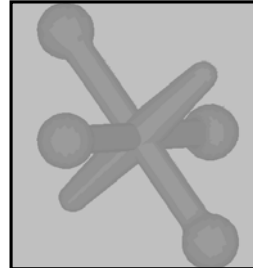
Oracle Spatial is an integrated set of functions and procedures that enables location data to be stored, indexed, accessed, and analyzed quickly and efficiently in an Oracle database.

Types of Location Data

GIS (mapping) data



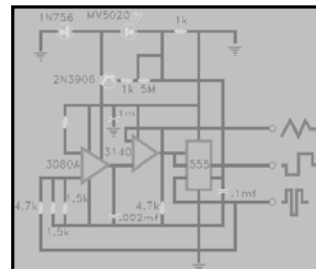
CAD data



Points of Interest



CAM data



ORACLE

2-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Introduction to Spatial Data

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 the data was collected, or land use code of an area.

Oracle Spatial is designed to make the storage, retrieval, and manipulation of spatial data easier and more natural to users. Once this data is stored in Oracle, it can be easily and meaningfully manipulated and retrieved as it relates to all the other data stored in the database.

The types of spatial data that can be stored using Oracle Spatial include location-based and geographic information system (GIS) data as well as data from computer-aided design (CAD) and computer-aided manufacturing (CAM) systems. Instead of operating on objects on a geographic scale, CAD and CAM systems work on a smaller scale such as an automobile engine, or a much smaller scale, such as printed circuit boards.

Introduction to Spatial Data (continued)

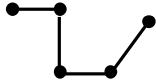
The differences among these three systems are only in the scale of the data, not its complexity. They might all involve the same number of data points. On a geographic scale, the location of a bridge can vary by a few tenths of an inch without causing any noticeable problems to the road builders. However,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

Geometric Primitive Types

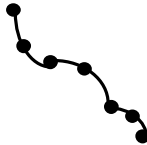
Point



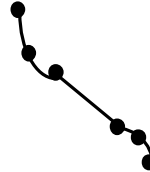
Line string



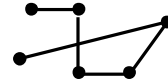
Arc line string



Compound line string

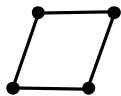


Self-crossing line strings



Valid

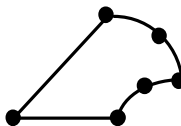
Polygon



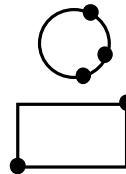
Polygon with hole



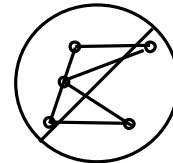
Compound polygon



Optimized polygons



Self-crossing polygons



Not valid

Geometric Primitive Types

Oracle Spatial supports many geometric primitive types. Each of these types will be discussed in detail in the following slides.

Geometric Primitive Types

- **Points (X1, Y1)**
 - Example uses include locations of buildings, customers, or ATMs
- **Line Strings (X1, Y1, Xn, Yn)**
 - Self-crossing lines are supported
 - Lines that close to form a ring have no area
 - Straight lines or circular arcs
 - Example uses include rivers, roads, cables, pipelines, and other linear features

ORACLE

2-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Geographic Primitive Types (continued)

Points

- A coordinate can be either 2D, 3D, or 4D.
- A point could represent a building, a customer location, or an ATM.

Lines Strings

- A series of two or more points that are connected by either straight lines or circular arcs. Line strings can cross themselves. Line strings have no area, even if they close to form a ring. As you will see in a future lesson, you can denote whether the line string is connected with straight lines or circular arcs.
- A line string could represent a river, a road, a utility cable, or any other linear feature.

Geometric Primitive Types

- **Polygons (X1, Y1, Xn, Yn)**
 - **Coordinates must close and area is implied**
 - **Interior area can be voided using additional polygons (for example, a polygon with holes)**
 - **Self-crossing polygons are not supported**
 - **Straight lines, circular arcs, optimized circle, rectangle**
 - **A polygon with no voids is considered an outer ring and must have counterclockwise rotation**
 - **A polygon can represent a land parcel, region, park, or any feature with area**

ORACLE

2-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Geographic Primitive Types (continued)

Polygons

- A polygon is a series of connected points, where the first point is the same as the last point (repeating the last point is an OGC standard requirement).
- Coordinates must close and interior is implied.
- Polygons can contain voids by using additional polygons that represent holes.
- Self-crossing polygons are not supported.
- Polygons can be made up of all straight lines or all circular arcs. Optimized circles and optimized rectangles are also categorized as polygons. An optimized circle (not really a polygon) is grouped with polygons because it has area. An optimized circle is represented by 3 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.
- A polygon with no voids is considered an outer ring and must have a counterclockwise rotation.
- A polygon can represent a land parcel, a region, a sales region, a park, and so on.

Geometric Primitive Types

- **Polygons (continued)**
 - Post 8.1.5, outer rings must be counterclockwise, inner rings (voids) must be clockwise
 - Post 8.1.5, when storing polygons with voids, outer rings must be followed by all its voids
 - Spatial migration tool takes polygons from 8.1.5 format (where polygon rotation and ordering of outer rings/voids did not matter) to post 8.1.5 requirements discussed above

ORACLE

2-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Geographic Primitive Types (continued)

Polygons (continued)

- Post 8.1.5, when storing polygons with voids, outer rings must be counterclockwise, and inner rings must be clockwise.
- Post 8.1.5, when storing polygons with voids, the outer ring must be first, and all the inner rings must follow the outer ring.
- A spatial migration tool is provided that fixes the rotation and ordering of polygons.

Geometric Primitive Types

- **Compound Line Strings (X1, Y1, Xn, Yn)**
 - Combination of straight lines and circular arcs
 - Self-crossing lines are supported
 - Lines that close to form ring have no area
- **Compound Polygons (X1, Y1, Xn, Yn)**
 - Combination of straight lines and circular arcs
 - Same semantics as polygons

ORACLE

2-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Geographic Primitive Types (continued)

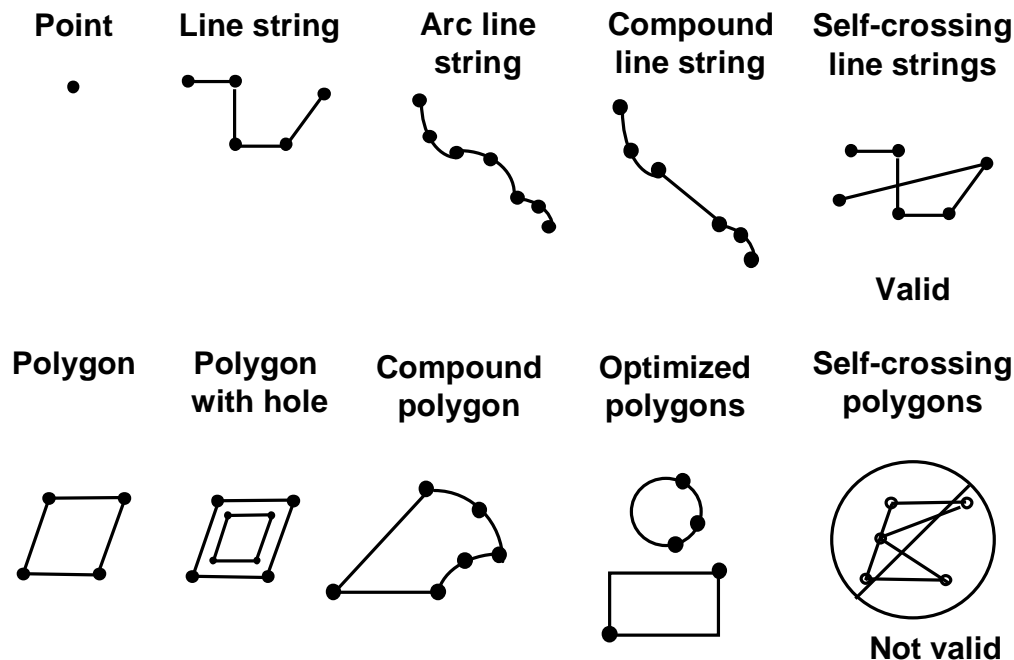
Compound Line Strings

- The line strings discussed earlier were either all connected with straight lines or all connected with circular arcs. Think of a compound as 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.
- Have the same properties of a line string. 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. Think of compound as combination. 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 which we will discuss in a future lesson. If you buffer a line, you will likely get a compound polygon (that is, straight lines on either side of the line you buffered and half circles around the end points).
- Same semantics as described for polygons (cannot self cross, rotation matters, and so on)

Geometric Primitive Types



2-10

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE

Supported Geometric Types

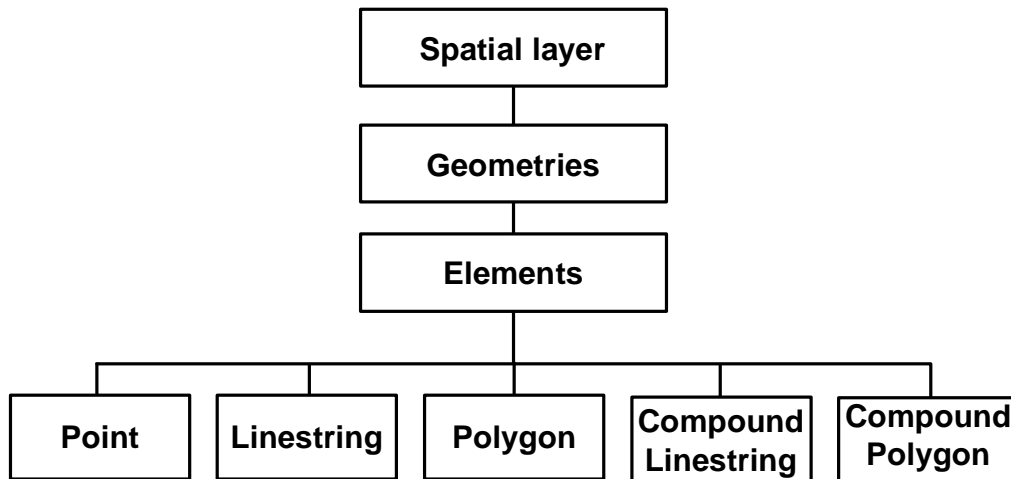
Points are elements composed of between two and four ordinates. Often, if only X and Y, they correspond to longitude and latitude.

Line strings are composed of two or more pairs of points that define line segments. Self-crossing lines are supported. Lines that cross to form a ring have no area.

Polygons are composed of line strings that form a closed ring, and the area of the polygon is implied. Polygons may also contain holes which are constructed by defining inner polygons. Holes represent polygon voids (doughnuts). 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.

Polygon can be exterior polygons or interior polygons (also called holes). An exterior polygon has to be followed by its interior polygons. Exterior polygons must be stored in counterclockwise rotation. Interior polygons must be stored in clockwise rotation. Interior polygons can only touch their exterior polygon at one point.

Spatial Data Model



ORACLE

2-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Data Model

The Oracle Spatial data model is a hierarchical structure.

Working from bottom to top, an element is synonymous to a geographic primitive type as discussed in prior slides. An element can be either a 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:

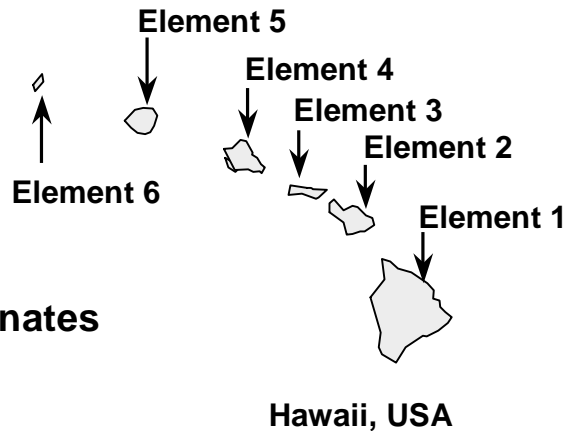
- 6th Avenue in New York City: 6th Avenue starts in southern Manhattan, hits Central Park, and continues on the other side of Central Park. This is an example of a geometry with 2 line string elements.
- U.S. State of Hawaii: The state of Hawaii is composed of multiple islands. This is an example of a geometry with several polygon elements.

The elements that make up a geometry do not all have to be the same element type. You can have a geometry that is made up of a point and polygon element.

A layer in Oracle Spatial is synonymous to a column in a table. A layer is made up of many geometries, each stored as a separate row. Data modeling is extremely important. You want to make sure when you define your table that contains a geometry column, that all the other columns in that table have a 1-to-1 relationship with that geometry. Typical Oracle data modeling concepts apply. Sets of geometries are stored in a spatial layer because they share common attributes.

Elements

- Basic building blocks of a geometry
- Same as geometric primitive types
- Element types
 - Point
 - Line
 - Polygon
 - Compound line string
 - Compound polygon
- Constructed using ordinates



ORACLE

2-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Element

Elements are the building blocks of a geometry and synonymous 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 above, Hawaii, USA, is a geometry made up of six polygon elements.

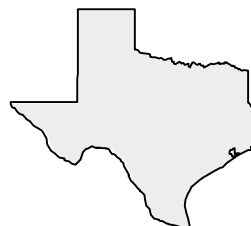
Geometry

- Represents a spatial feature
- Consists of an ordered set of elements

**Geometry 1
California**



**Geometry 2
Texas**



**Geometry 3
Florida**



**Geometry 4
Hawaii**



ORACLE

2-13

Copyright © Oracle Corporation, 2001. All rights reserved.

Geometry

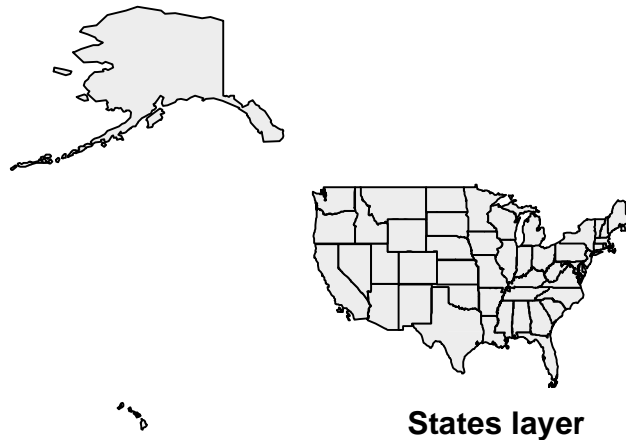
A geometry is the representation of a user's spatial feature, modeled as an ordered set of geometric primitive elements.

Each of the geometry examples above are examples of U.S. states (California, Texas, Florida, Hawaii) that are made up of multiple elements, where all of the elements are the same type. Geometries that contain different element types are supported by Oracle Spatial.

A polygon with holes is stored as a sequence of polygon elements inside of a geometry. See the Geometric Primitive Types slide for polygons for the rules governing polygons with holes in Oracle Spatial.

Layers

- **Consist of geometries that share a common set of attributes**
- **Layer is a geometry column in a table**



Layers

A layer is synonymous with a column in a table. A future lesson will discuss that there is a new native data type in Oracle. Just as you have NUMBER, VARCHAR2, and DATE, you now have a new native data type called MDSYS.SDO_GEOMETRY. An Oracle table can have more than one SDO_GEOMETRY column (in other words, more than one layer). For instance, a table might contain all of the states in the United States at a low resolution, and all of the states at a high resolution. The application can determine which layer to query depending on the zoom level.

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.

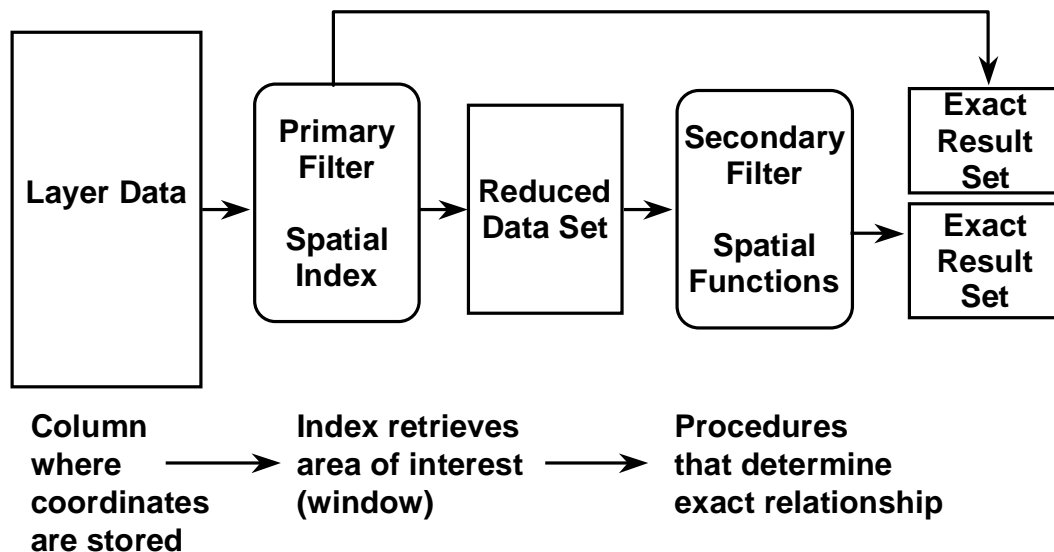
Data modeling is very important, but is no different than ordinary Oracle data modeling. You want to make sure the SDO_GEOMETRY column in your table has a 1-to-1 relationship with the other columns in that table. This shouldn't be a new concept for anyone who knows how to create a normalized table.

Layers (continued)

For example, even though roads and rivers are both features that can be represented with line strings, you should not store rivers in the same SDO_GEOMETRY column as roads. Roads and rivers do not share a common set of attributes (for example, roads may have a property called lanes which mean nothing to a river, and rivers may have a property called salinity, which means nothing to a road). 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.

The States layer shown in the slide could be modeled in a table that contained 50 rows, one for each U.S. state.

Optimized Query Model



ORACLE

2-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Optimized Query Model

This slide describes the Oracle Spatial optimized query model. It probably will not entirely make sense the first time you see it. But this slide will be repeated several times throughout this course, and each time you see it, it will make more sense.

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 you create a spatial index, the spatial index stores geometry approximations for each geometry. The primary filter portion of the query compares the geometry approximations instead of the true geometries, and generates a reduced data set. Comparing geometry approximations is much faster than comparing the true geometries. There are some optimizations Oracle does in the primary filter, where it can accept geometries as truly interacting, even though only geometry approximations are compared. This is why you see some results going to the exact result set, and some going to the reduced data set.

Optimized Query Model (continued)

Oracle Spatial gives you the option of just performing a primary filter. A primary filter will give you all the geometries that satisfy a query, and possibly some additional geometries. Some applications do not need an exact answer, for example, ZOOM IN/OUT on a mapping viewport. The ZOOM IN/OUT can perform a primary filter, and use its clipping routines to discard the additional geometries that returned from the primary filter.

If the exact answer is desired, for example, if you drew a circle on the viewport and asked for all the parcels that interacted with the circle, you probably do not want a fuzzy answer (that is, a few more parcels than the true answer). For the exact answer, you can run a secondary filter, which compares only the coordinates of the geometries that were returned in the primary filter that were not automatically accepted as exact, not the entire layer. The geometries returned in this exact comparison added to the geometries accepted as exact in the primary filter stage will result in the exact answer.

Spatial Indexing

- **Two basic types**
 - **Quadtrees**
 - Based on a quadtree decomposition**
 - Indexes two dimensions**
 - **R-trees**
 - Based on minimum bounding rectangles (MBRs)**
 - Indexes two, three, or four dimensions**
- **Provides an exclusive and exhaustive coverage of spatial objects**
- **Indexes points, lines, and polygons**

ORACLE

2-18

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Indexing

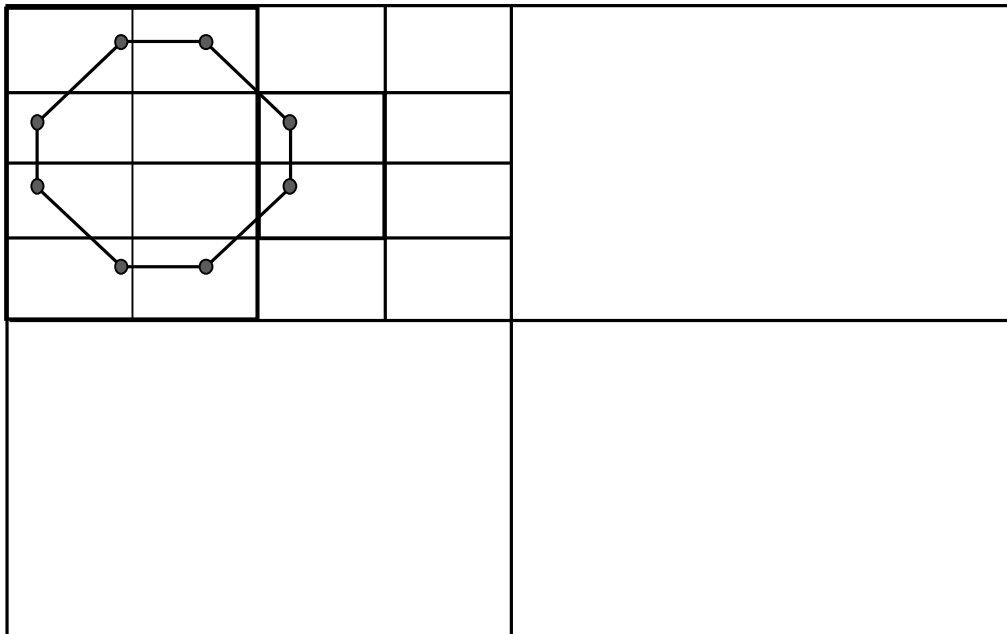
Oracle Spatial uses a spatial index to optimize spatial query performance. In the same way an index is created on scalar data to improve query performance, a spatial index improves spatial query performance. There are two supported index types, quadtree indexes and R-tree indexes (R-trees are in production as of Oracle 8.1.7). There are scenarios where quadtrees perform better than R-trees and scenarios where R-trees perform better than quadtrees.

Because R-tree indexes require almost no tuning, and quadtree indexes require lots of tuning, Oracle's goal is to bridge the gap of scenarios where the R-tree index does not perform as well as the quadtree index.

In Oracle9i Release 1, Oracle has made a big step at bridging the gap between R-trees and quadtrees. In most scenarios, the R-tree index will perform as well as the quadtree or better. Additional R-tree optimizations are planned in Oracle 9i Release 2.

Some new functionality introduced in Oracle9i will only work with R-tree indexes (for example, incremental nearest neighbor, and some queries using the Whole Earth geometry model). Because R-trees require almost no tuning, most of the indexing discussions and tuning exercises in this course will focus on quadtree indexes. Regardless of which index type you choose (R-tree or quadtree), Oracle generates and stores a rough 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).

Quadtree Decomposition of Space



ORACLE

2-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Tiles

The example in this slide demonstrates the quadtree indexing concept. For quadtrees, you pick a tile size, and that sized tile will be used to generate a grid of tiles that approximates the geometry.

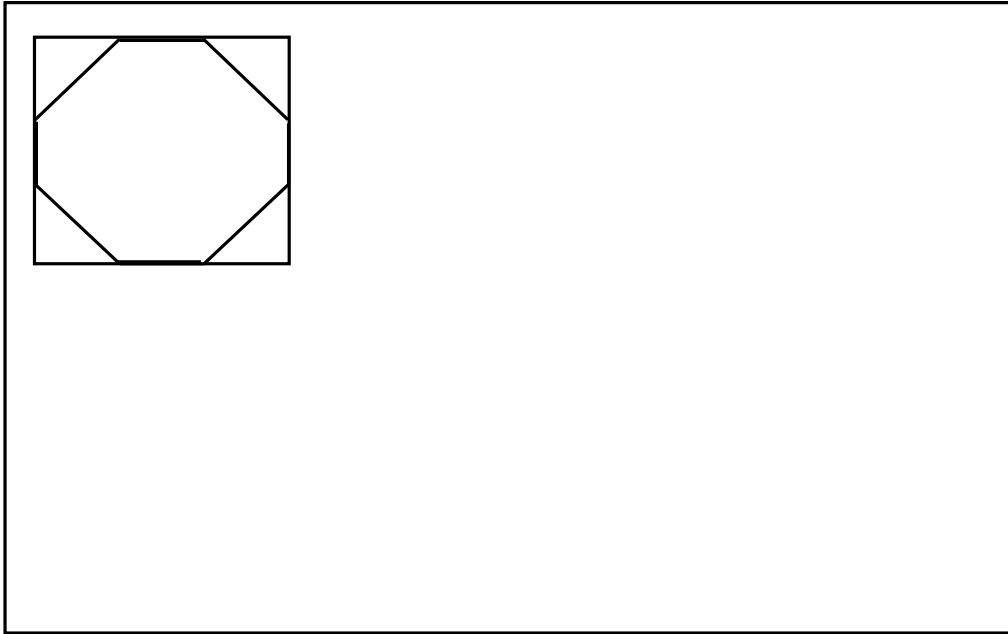
The tile size is directly related to a parameter of the create index statement called `SDO_LEVEL`. `SDO_LEVEL` is only applicable for generating quadtree indexes.

An `SDO_LEVEL=0` sized tile represents the rectangle that corresponds to the entire coordinate system extent. Quadtrees always decompose each tile at the previous level into four tiles at the next level. An `SDO_LEVEL=1` sized tile is 1/4 the size of an `SDO_LEVEL=0` tile. An `SDO_LEVEL=2` tile is 1/4 the size of an `SDO_LEVEL=1` sized tile, etc.

In the example above, `SDO_LEVEL=3` sized tiles were used. This may not be the most optimal `SDO_LEVEL` to use, but it is a good `SDO_LEVEL` to demonstrate the quadtree indexing concept.

The tuning lesson will go into detail on how to pick an optimal `SDO_LEVEL` for a quadtree fixed index.

R-tree: Generates Minimum Bounding Rectangle (MBR) Tile



ORACLE

2-20

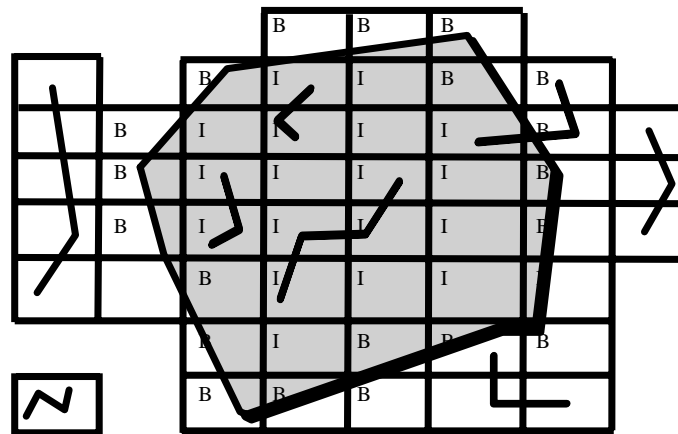
Copyright © Oracle Corporation, 2001. All rights reserved.

Tiles (continued)

The example in this slide demonstrates the R-tree indexing concept. For each geometry, a single minimum bounding rectangle tile will be generated during index creation.

No tile size tuning is necessary.

Primary and Secondary Filter Concept Quadtree Example



Primary and Secondary Filter Concept

Say there are 20 million roads (line strings) in a layer and a query is run to determine which roads interact with a zip code polygon. If there were no spatial index, computational geometry to compare each line segment of each road to the perimeter and interior of the polygon to see if there is any interaction would be required. This would require a full scan of all the roads.

An important task of a spatial index is to reduce the number geometries that need to be compared. Comparing the line segments and interiors of geometries is a very expensive process with respect to time. The primary filter only compares tiles, not the true geometries. In this example, a quadtree spatial index was used. For each road, quadtree tiles were generated to approximate the road. Quadtree tiles were also generated to approximate the polygon.

Because only approximations are compared in the primary filter, it will return a superset of the true result. It returns the entire answer, and possibly a few more geometries. The primary filter will return all the roads that have a tile that is the same as one of the tiles that approximates the polygon. In this case, five yellow roads returned in the primary filter (four that truly interact, and one that doesn't).

Primary and Secondary Filter Concept (continued)

Oracle Spatial gives you the option of just performing a primary filter. A primary filter will give you all the geometries that satisfy the query, and possibly a few additional. Some applications do not need an exact answer, for example, ZOOM IN/OUT on a mapping viewport. The ZOOM IN/OUT can perform a primary filter, and use its clipping routines to discard the additional geometries that returned from the primary filter.

If the exact answer is desired you can run a secondary filter, which only compares the coordinates of geometries that returned in the primary filter, not all 20 million roads. This will result in the exact answer.

In addition, Oracle Spatial applies an interior tile optimization to further improve secondary filter performance. Tiles for the polygon are tagged as interior (I) or boundary (B). If a road tile equals the polygon's I tile, that geometry can get accepted as having passed the secondary filter during the primary filter process (no need to compare the line segments of the geometries). This optimization is done for all secondary filter masks (for example, TOUCH relationships will only evaluate roads that touch B tiles).

A similar interior tile optimization is also done with R-trees. Quadtree indexes require tuning for the interior tile optimization. This optimization is transparent with R-tree indexes, and is not part of the tuning process.

Spatial Window Queries

- **Issued through SQL**
- **Contains a location constraint, for example:**
 - Find all policy holders in the projected path of a hurricane
 - Find all ATMs in my area

ORACLE

2-23

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Window Queries

All spatial queries are issued through SQL.

The term “window” and “area-of-interest (AOI)” are used synonymously in this course. Window refers to the area you are searching.

The extreme case of a window query is a single window against an entire layer. Some examples are given above:

- The hurricane path is a single window used to query the policy holders layer.
- Your area could be defined as anywhere within two miles from where you are (for example, within a two mile radius). That window is used to query the ATMs layer.
- Certainly you can have window queries that have more than a single window (for example, show all the ATMs in region a or region b). More about how to implement spatial queries will be discussed in the “Spatial Queries” lesson.

Spatial Joins (Spatial Cross Product)

- **A join of two layers based on the spatial component of the layers**
- **Implemented using spatial index and spatial operators**
- **Very rare**
- **Example:**
 - **Find all the wetlands within the national parks**

ORACLE

2-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Joins

You can think of a spatial join as a spatial cross product. It compares all of the geometries in one layer to all of the geometries in another layer. The example above is comparing all the wetlands to all the national parks to find all the national parks that have wetlands.

For the most part, spatial joins are a rare requirement. But there are scenarios where you would want to compare all the geometries from one layer to all the geometries in another layer. For example, if you have a county layer and a customer layer, and want to generate a report that identifies how many customers you have in each county, then in this example you would be comparing all the customers to all the counties.

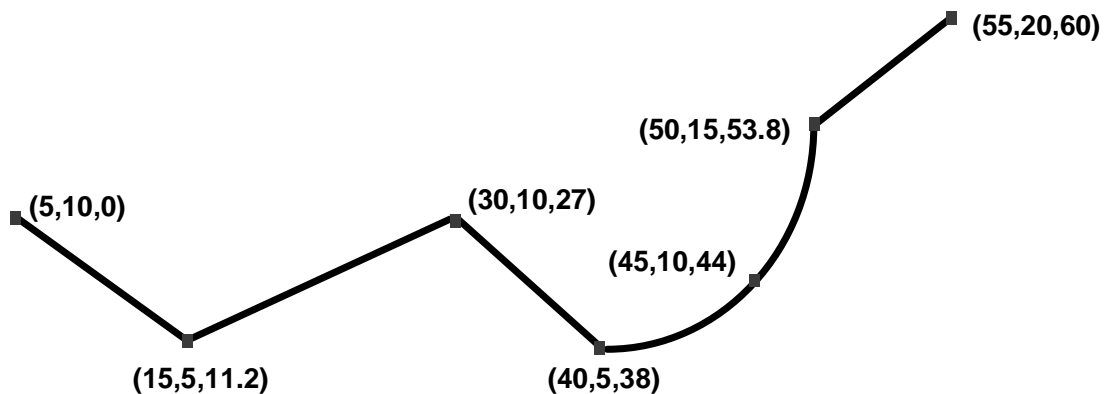
Even though you can accomplish this with a JOIN query, Oracle recommends you do this as a WINDOW query that iterates through each window. The interior tile optimization discussed earlier is never applied for JOIN queries.

By using WINDOW queries instead of JOIN queries, you can greatly reduce the true geometry comparisons in the secondary filter operation by utilizing the interior tile optimization.

In the “Spatial Query” lesson, you will learn how to write any JOIN query as a WINDOW query.

Linear Referencing System (LRS)

- A mechanism to associate a measure value with a 2D or 3D point along a line string or a polygon
- Measure value is typically proportional to the distance from the start measure of the geometry
- LRS will be discussed in more detail in a future lesson



ORACLE

2-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Linear Referencing System (LRS)

Linear referencing is commonly used by Departments of Transportation. A common usage is to assign mileposts on a highway. In another table, you can associate measure ranges to nonlinear values like a speed limit, or road condition. Speed limit is not a linear value; it's a discrete value. It doesn't increase the further you traverse a road as mileposts do.

Linear referencing systems associate a measure value with each 2D or 3D point along a geometry. In the example above, 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 point will be between 0 and 60. Measure values can be ascending (as in the example above) or descending.

Your linear referenced geometries are stored in one table, and in another table you can associate measure value ranges with discrete values like speed limits. For example, for the road depicted in this slide, measure values 0 through 27 could be a 50 miles per hour zone, and measure values 27 through 60 could be 35 miles per hour zone.

Summary

In this lesson, you should have learned about:

- **The supported geometric primitive types**
- **The Oracle9i spatial data model**
- **Spatial layers**
- **The spatial query model**
- **The concept of spatial indexing**
- **Spatial window queries and spatial joins**
- **Linear Referencing System (LRS)**

ORACLE

3

Creating Spatial Layers

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Describe the schema associated with a spatial layer**
- **Explain how spatial data is stored using the Oracle Spatial object-relational model**
- **Create a table with a spatial layer**
- **Load the metadata view associated with a spatial layer**

ORACLE

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**

```
SQL> CREATE TABLE states (  
2     state      VARCHAR2(30),  
3     totpop     NUMBER(9),  
4     geom       MDSYS.SDO_GEOMETRY);
```

ORACLE

3-3

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOMETRY Object

In Oracle8i a new “native” data type, SDO_GEOMETRY, was introduced into Oracle. You create tables with columns of SDO_GEOMETRY type in the same way 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 SDO_GEOMETRY type. It is an ordinary Oracle table, with ordinary storage clauses, that now contains one or more SDO_GEOMETRY columns. The SDO_GEOMETRY object will contain an entire geometry in a single row, single column of an Oracle table.

If you know how to use Oracle, and never heard of Oracle Spatial, you know how to use 90% of the product. You can leverage the knowledge of the existing DBAs within your enterprise. The 10% new information you will learn in this course deals with how to load, index, and query an SDO_GEOMETRY column. The SDO_GEOMETRY column is accessed through SQL, and all of the same utilities (SQL*Loader, Import, Export) are used. The SDO_GEOMETRY object has five fields. Each will be discussed in detail in the next few slides. Two of the fields are of the NUMBER type and the other three fields are object types.

SDO_GEOMETRY Object (continued)

Above is a simple example of creating an ordinary Oracle table called STATES with an SDO_GEOMETRY column. Any column that has a 1-to-1 relationship with the geometry column should be placed in the table (ordinary data modeling concepts for normalized tables).

Note: The SDO_GEOMETRY object is owned by the Oracle MDSYS user. This is a privileged Oracle user with all privileges. You should treat MDSYS as a system user, and never create data or objects in the MDSYS Oracle schema. Also, by default, the MDSYS password is MDSYS. You should change this password so hackers will not gain inappropriate access.

SDO_GEOMETRY Object

- **SDO_POINT_TYPE**

| | |
|----------|---------------|
| x | NUMBER |
| y | NUMBER |
| z | NUMBER |

- **SDO_ELEM_INFO_ARRAY**

VARRAY (1048576) OF NUMBER

- **SDO_ORDINATE_ARRAY**

VARRAY (1048576) OF NUMBER

ORACLE

3-5

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOMETRY Object (continued)

The SDO_POINT_TYPE object is a field within the SDO_GEOMETRY object, and should not be used outside of 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 one or more elements stored in this geometry. This field will be discussed in detail in an upcoming slide.

SDO_ORDINATE_ARRAY is a field within the SDO_GEOMETRY object. It will contain all the ordinates for all the elements that make up the geometry. This will be described in more detail in upcoming slides.

SDO_GEOMETRY Object

- **SDO_GTYPE**
- **Defines the type of geometry stored in the object**

| GTYPE | <u>Explanation</u> |
|-----------------------------------|---|
| 0 UNKNOWN_GEOMETRY | Spatial ignores this geometry. |
| 1 POINT | Geometry contains one point. |
| 2 LINESTRING | Geometry contains one line string. |
| 3 POLYGON | Geometry contains one polygon. |
| 4 HETEROGENEOUS COLLECTION | Geometry is a collection of elements of different types: points, lines, polygons |
| 5 MULTIPOINT | Geometry has multiple points. |
| 6 MULTILINESTRING | Geometry has multiple line strings. |
| 7 MULTIPOLYGON | Geometry has multiple polygons. |

ORACLE

3-6

Copyright © Oracle Corporation, 2001. All rights reserved.

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. Think of the SDO_GTYPE as the big picture that is, what kind of geometry is being stored in the SDO_GEOMETRY object.

SDO_GTYPE 0: The geometry in this row will be ignored by Oracle Spatial. SDO_GTYPE 0 is almost never used.

SDO_GTYPE 1: The geometry contains a single point element.

SDO_GTYPE 2: The geometry contains a single line string element.

SDO_GTYPE 3: The geometry contains a single polygon element with zero or more voids.

SDO_GTYPE 4: The geometry contains one or more elements, and all the elements can be of different types.

SDO_GTYPE 5: The geometry contains one or more point elements.

SDO_GTYPE 6: The geometry contains one or more line string elements.

SDO_GTYPE 7: The geometry contains one or more polygon elements which may or may not contain voids.

Try to associate the most appropriate SDO_GTYPE with your geometry. For example, a single polygon element could be categorized as an SDO_GTYPE 7, but optimally it will be categorized as an SDO_GTYPE 3.

SDO_GTYPES

| GTYPE | Post 8.1.5 GTYPES - Determine Dimensionality | | |
|--------------------|--|-----------|-----------|
| | <u>2D</u> | <u>3D</u> | <u>4D</u> |
| 0 UNKNOWN_GEOMETRY | 2000 | 3000 | 4000 |
| 1 POINT | 2001 | 3001 | 4001 |
| 2 LINESTRING | 2002 | 3002 | 4002 |
| 3 POLYGON | 2003 | 3003 | 4003 |
| 4 COLLECTION | 2004 | 3004 | 4004 |
| 5 MULTIPOINT | 2005 | 3005 | 4005 |
| 6 MULTILINESTRING | 2006 | 3006 | 4006 |
| 7 MULTIPOLYGON | 2007 | 3007 | 4007 |

- In 8.1.5, determining the dimensionality of a geometry required a query to the SDO_GEOM_METADATA table
- 8.1.5 style SDO_GTYPES still supported for backward compatibility, although some new functionality requires migrated data
- Post 8.1.5, Oracle Spatial includes a tool to migrate data to the new SDO_GTYPES

ORACLE

3-7

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOMETRY Object: SDO_GTYPES

In Oracle 8.1.5, the SDO_GTYPE was a single digit (0 through 7). Post Oracle 8.1.5, a four digit SDO_GTYPE was introduced. The dimensionality of the geometry is now also encoded into the SDO_GTYPE as the first of the four digits. The dimensionality determines how many numbers make up a point (that is, 2, 3, or 4 numbers).

Prior to storing the dimensionality in the SDO_GTYPE, the only way to get the dimensionality of a geometry was to query a Spatial dictionary table to see how many axis were defined for the geometry column (that is, do 2, 3, or 4 numbers make up a point). The Spatial dictionary table's primary key is (table_name, column_name), and for each Spatial column, the axis definitions are defined. Adding the dimensionality to the SDO_GTYPE made it possible to eliminate the Spatial dictionary search to determine a geometry's dimensionality.

For example, imagine you create a view which is a join of multiple tables, where one of the selectable columns in the view is a geometry column. Determining the table and column name from which the geometry was derived required a very complex SQL parser for the view.

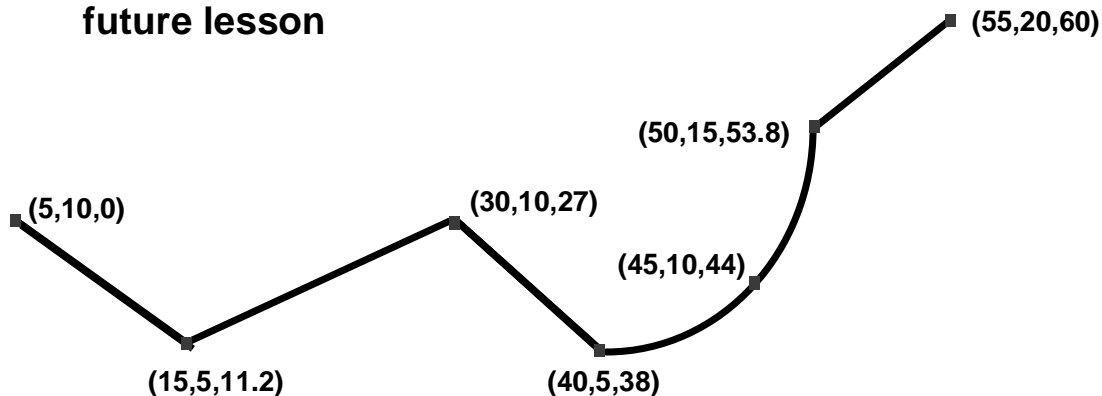
SDO_GEOMETRY Object: SDO_GTYPEs (continued)

To avoid this difficult scenario, Oracle Spatial (after 8.1.5) stores the dimensionality in the first digit of the SDO_GTYPE. This enables you to determine the dimensionality of the geometry without a dictionary lookup.

The single digit SDO_GTYPE is still supported for backward compatibility although some new features require migrated data. An Oracle Spatial migration utility (SDO_MIGRATE.TO_CURRENT) will migrate spatial data to the four digit SDO_GTYPEs.

Linear Referencing System (LRS)

- A mechanism to associate a measure value with a 2D or 3D point along a line string, multiline string, or polygon
- Measure value is typically proportional to the distance from the start measure of the geometry
- LRS will be examined in more detail in a future lesson



ORACLE

3-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Linear Referencing System (LRS)

Linear referencing systems associate a measure value with each 2D or 3D point along a geometry. In the example above, 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 point will be between 0 and 60. Measure values must be either ascending (as in the example above) or descending.

Linear-referenced geometries are stored in one table, and in another table are associated measure value ranges with discrete attributes like speed limits. 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 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 above, 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.

If you set a measure value for an intermediate point, and other measure values in the geometry are set to NULL, Oracle Spatial LRS has a function that will interpolate the geometry NULL measure values, taking into account any measure values you have set in the geometry.

SDO_GTYPE with LRS: Oracle9i

| GTYPE | GTYPES - Include dimensionality and LRS dimension | | |
|--------------------|---|------|------|
| | 2D | 3D | 4D |
| 0 UNKNOWN_GEOMETRY | 2000 | 3000 | 4000 |
| 1 POINT | 2001 | 3a01 | 4b01 |
| 2 LINESTRING | 2002 | 3a02 | 4b02 |
| 3 POLYGON | 2003 | 3a03 | 4b03 |
| 4 COLLECTION | 2004 | 3004 | 4004 |
| 5 MULTIPOINT | 2005 | 3005 | 4005 |
| 6 MULTILINESTRING | 2006 | 3a06 | 4b06 |
| 7 MULTIPOLYGON | 2007 | 3007 | 4007 |

a = 0 - No LRS or not known as such
a = 3 - LRS measure is 3rd dimension
b = 0 - No LRS or not known as such
b = 3 - LRS measure is 3rd dimension
b = 4 - LRS measure is 4th dimension

ORACLE

3-10

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GTYPE with LRS: Oracle9i and Beyond

In Oracle9i, the position of the measure in an LRS geometry is encoded in the SDO_GTYPE in the second digit.

If the geometry is not an LRS geometry, then there is no change to the SDO_GTYPE (that is, 2000 - 2007, 3000 - 3007, and 4000 - 4007).

If the geometry is an LRS geometry, then it can be either:

- 2D geometry with a measure: A geometry with x,y values and a measure for each point. In this scenario, the measure must always be in position 3 (that is, SDO_GTYPE 3301, 3302, 3303, or 3306)
- 2D geometry with a measure and another non-Z value: A geometry with x,y values, a measure, and some other values (not z) stored for each point. In this scenario, the measure can be in position 3 or 4 (that is, SDO_GTYPE 4301, 4302, 4303, 4306, 4401, 4402, 4403, or 4406).
- 3D geometry with a measure: A geometry with x,y,z values (z is height) and a measure for each point. In this scenario, the measure must always be in position 4 (that is, SDO_GTYPE 4401, 4402, 4403, or 4406).

The LRS position must be consistent in all of the geometries in the column.

Why are the measure positions stored in the geometry? The same reason the dimensionality is stored in the geometry. Storing this information in the the geometry eliminates a dictionary lookup to find the measure position, which is especially difficult if the geometry comes from a complex view (as discussed previously in the notes of the SDO_GTYPE slide).

LRS Measure Positions: Summary

- **2D geometry with a measure**
 - 2D geometries with a measure must have the measure value in the third position of each point.
- **2D geometry with a measure, and another non- Z value**
 - 2D geometries with a non-Z value and a measure can store the measure in either the third or fourth position of each point. The measure position must be consistent for all the points in the geometry.
- **3D geometry with a measure**
 - 3D geometries (that is, x,y,z values) and a measure must store the measure value in the fourth position of each point.

ORACLE

3-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary of LRS Measure Positions

Two-dimensional geometries with measure must always have the measure value stored in the third position (or dimension) of each coordinate.

Two-dimensional geometries with a measure and another non-Z value can store the measure value in either the third or fourth position of each coordinate. The measure position has to be consistent for all measures in the geometry as well as all geometries in the layer.

Three dimensional geometries with a measure must always store the measure value in the fourth position of each point defined in the geometry.

New Methods on SDO_GTYPE

- **GET_DIMS()** returns number of dimensions

```
SQL> select s.geom.get_dims() state_dims
       from states s
       where s.state = 'Delaware';
STATE_DIMS
-----
          2
```

- **GET_GTYPE()** returns geometry type

```
SQL> select s.geom.get_gtype() state_gtype
       from states s
       where s.state = 'Delaware';
STATE_GTYPE
-----
          3
```

ORACLE

New Methods on SDO_GTYPE

There are new methods in Oracle9i that are available to parse information out of the SDO_GTYPE.

GET_DIMS() returns the dimensionality of the geometry

GET_GTYPE() returns the GTYPE that describes what type of geometry is being stored.

New Methods on SDO_GTYPE

- **GET_LRS_DIM()** returns LRS dimension

```
SQL> select s.geom.get_lrs_dim() state_lrs_dim
       from states s
       where s.state = 'Delaware';
```

```
STATE_LRS_DIM
```

```
-----
```

```
0
```

ORACLE

New Methods on SDO_GTYPE (continued)

GET_LRS_DIM() returns the LRS position of the geometry (the query returns 0 if the geometry is not known to be linear referenced).

Note: Sometimes it is better to parse the SDO_GTYPE in your application instead of using the GET_DIMS and GET_LRS_DIM methods. Both of these methods will return the same value for every geometry in the layer. For example, every geometry in a layer must have the same dimensionality, hence GET_DIMS will return the same value for every row in a layer. To reduce network traffic (that is, less columns in your select list), parse the SDO_GTYPE in your application to get dimensionality information instead of calling GET_DIMS or GET_LRS_DIM in the select list.

SDO_GEOMETRY Object

SDO_SRID

- **Spatial Reference System ID**
- **Used for coordinate systems support**

SDO_POINT

- **Optimized space for storing points (not point clusters)**
- **Ignored if SDO_ELEM_INFO and SDO_ORDINATES are not NULL**
- **SDO_POINT_TYPE should not be used outside of the SDO_GEOMETRY object**

ORACLE

3-14

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOMETRY Object

Up to now, we have only discussed the first of 5 fields in the SDO_GEOMETRY object. The second field is SDO_SRID.

SDO_SRID field

This is the second field in the SDO_GEOMETRY object. This is how you associate your data with a coordinate system. Oracle Spatial has a dictionary table called MDSYS.CS_SRS which lists all the supported coordinate systems. The value you set the SDO_SRID field must match a value in the SRID column in the MDSYS.CS_SRS table, or be NULL. All geometries in a column must have the same SDO_SRID. We will go into more detail about coordinate systems in the coordinate system lessons.

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. The use of the SDO_POINT field must be within the context of the SDO_GEOMETRY object (Oracle Spatial will not work with a column in a table defined as SDO_POINT_TYPE). If you want to store a single two or three dimensional point, store it in the SDO_POINT field of the SDO_GEOMETRY object. Using the SDO_POINT field provides for optimal storage and performance. This field is ignored if the SDO_ELEM_INFO and SDO_ORDINATES fields are not null.

SDO_GEOMETRY Object

SDO_ELEM_INFO

- Entries in the array should be considered in groups of three
- The triplet values stored in the array are interpreted as:
 - Ordinate offset: The position of the first ordinate of an element in the SDO_ORDINATES array
Values start at 1
 - Element type
 - Interpretation: Straight line, Circular arc, or header

ORACLE

3-15

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_ELEM_INFO field

This is the fourth field in the SDO_GEOMETRY object. A geometry can contain one or more elements. If the geometry is not a single point stored in the SDO_POINT field, a description of each element that makes up a geometry is stored in the SDO_ELEM_INFO field, as described below.

The SDO_ELEM_INFO field is a VARRAY (varying sized array) of NUMBER. Entries in the SDO_ELEM_INFO field should be considered in groups of three numbers. The triplet values represent:

Ordinate offset: Because a geometry may be made up of one or more elements, this is an index into SDO_ORDINATES field described below. The index value corresponds to the first ordinate of an element. The first element of a geometry will always begin at position 1.

Element type: This is the element type (point, line, polygon, and others). The element type is also called the etype. We will discuss all possible values for the element type in the remaining portion of this lesson.

Interpretation: Interpretation takes on a different meaning depending on the value of element type. It can mean all points for this element are connected with straight lines, circular arcs, or it might mean this value is a header for a compound element. This will all be discussed in detail in upcoming slides.

SDO_GEOMETRY Object

SDO_ORDINATES

- **This is a simple VARRAY of numbers**
- **Contains the ordinates that make up geometry elements**
- **Elements stored in the SDO_ORDINATE_ARRAY are defined in the SDO_ELEM_INFO_ARRAY**

ORACLE

3-16

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_ORDINATES field

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

Element Example: Point

• (x, y, z)

- Load the SDO_POINT field to optimize point data storage
- A quadtree spatial index is two dimensional, x and y
- An R-tree spatial index indexes up to four dimensions
- 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)

```
SQL> INSERT INTO TELEPHONE_POLES
2>   VALUES (attribute_1, ..., attribute_n,
3>           MDSYS.SDO_GEOMETRY (
4>             3001, null,
5>             MDSYS.SDO_POINT_TYPE (-75.2,43.7,200),
6>             null, null)
7>   );
```

ORACLE

3-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example

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

If you use a quadtree spatial index, the spatial index will be 2D (that is, only x and y are indexed). Quadtree primary filters are 2D. If you use an R-tree spatial index, you can spatially index up to 4 dimensions. R-tree primary filters can be up to 4D. Regardless of which index type you choose, Oracle Spatial's secondary filters are always 2D (that is, volumetric comparisons are not supported). If you have a volumetric algorithm for comparing geometries, you can implement this as a PL/SQL or Java stored procedure, and call it on the results of an R-tree primary filter (in the same SQL statement). We will discuss spatial indexes, primary and secondary filters in more detail when we get to the spatial index and query lessons.

To generate a spatial index on 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.

Element Example (continued)

The above example shows the use of constructor (in red), which is syntax specific to object types. It can be thought of as a logical cast operation, forcing the data to the type specified. There are two constructors shown above, one for the SDO_GEOMETRY type and one inside of that for the SDO_POINT_TYPE.

Note: If you are storing a 2D point in the SDO_POINT field, set the z value to NULL; for example:

```
SQL> INSERT INTO TELEPHONE_POLES VALUES (attribute_1, attribute_n,  
2> MDSYS.SDO_GEOMETRY (2001, null, MDSYS.SDO_POINT_TYPE  
3> (-75.2,43.7,NULL), null, null));
```


Element Types Summarized

| <u>Number</u> | <u>Element Type</u> | <u>Interpretation</u> |
|---------------|---------------------|--|
| 0 | UNKNOWN_ELEMENT | |
| 1 | POINT | # of points in collection |
| 2 | LINESTRING | 1 - Straight lines 2 - Circular arcs |
| 3 | POLYGON | 1 - Straight lines |
| 1003 | (Outer) | 2 - Circular arcs |
| 2003 | (Inner) | 3 - Optimized rectangle |
| | | 4 - Circle |
| 4 | COMPOUND LINESTRING | # of type 2 subelements that make up the line string |
| 5 | COMPOUND POLYGON | # of type 2 subelements that make up the polygon |
| 1005 | (Outer) | |
| 2005 | (Inner) | |

ORACLE

3-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Types Summarized

So far, you have only seen how to store data in the SDO_POINT field of the SDO_GEOMETRY object. Now you will 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 will be repeated after the element examples, and will probably make much more sense 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 will have a triplet value entry in the SDO_ELEM_INFO (ordinate offset, element type, interpretation).

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

Don't confuse element types with SDO_GTYPEs. SDO_GTYPEs are the big picture. SDO_GTYPEs represent what type of geometry is stored. A geometry is made up of one or more elements, and the element type describes the element.

Element Types Summarized (continued)

In Oracle 8.1.5, the element types were all one digit (0-5). Post 8.1.5, a four digit element type for polygon elements was introduced. Element type 3 is replaced with either 1003 (outer ring) or 2003 (void), and 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 if a polygon element is an outer ring or a void.

Element type 0: Used to model an element that is not currently supported by Oracle Spatial. An example will be demonstrated in a future slide.

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: Post 8.1.5, 1003 represents an outer ring polygon, 2003 represents an inner ring (or void).

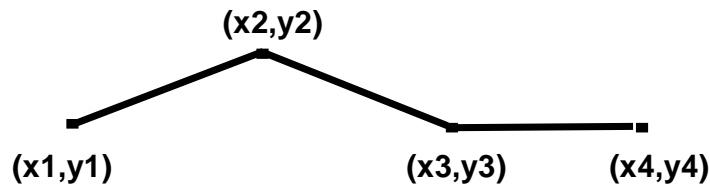
- Interpretation 1: Conventional polygon connected with straight lines
- Interpretation 2: Arc polygon, connected with circular arcs
- Interpretation 3: Optimized rectangle, only requires lower left and upper right points to represent the rectangle.
- Interpretation 4: Optimized circle (not truly a polygon, but categorized with polygons because it has area)

Element type 4: Compound line string, 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 (more details to come, with an example)

Element type 5: Post 8.1.5, 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 (more details to come, with an example).

Element Example: Line String

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 2 | 1 |



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

ORACLE

3-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Line String

So far, you have only been introduced to storing a single point in the SDO_POINT field of the SDO_GEOMETRY object. This slide is the first example showing what goes into the SDO_ELEM_INFO field of the SDO_GEOMETRY object.

In the boxes above is the triplet value (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field. Most of the ordinate offsets you will see in the upcoming examples start at 1 because the examples depict geometries with a single element. There will be some examples with geometries that have more than one element, and in those cases you will see ordinate offsets greater than 1.

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

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

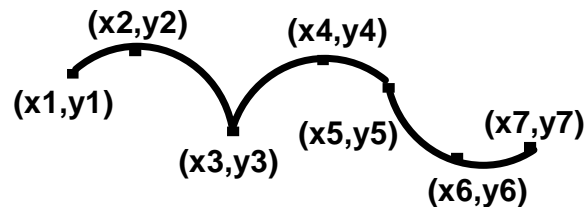
Line strings that close to form a ring have no implied area.

Line segments can cross each other.

Note: The SDO_GTYPE for this geometry is 2002 (single line string element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Element Example: Arc String

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 2 | 2 |



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

ORACLE

3-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Arc String

In the boxes above is the triplet value (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field.

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

The interpretation field for an element type 2 will define how to connect the segments that make up the line string. In this example, the interpretation is 2, and 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.

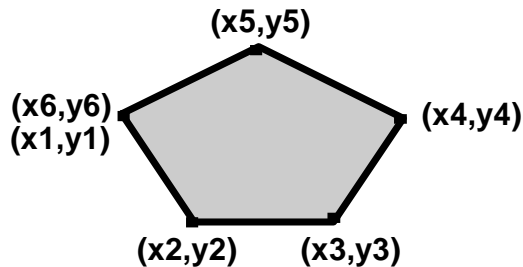
Arc strings that close to form a ring have no implied area.

Arcs can cross each other.

Note: The SDO_GTYPE for this geometry is 2002 (single line string element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Element Example: Polygon

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1003 | 1 |



- Interpretation 1: All line segments are straight lines
- Area is implied
- Line segments cannot cross each other
- Last coordinate equals first coordinate
- Order (rotation) matters

ORACLE

3-23

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Polygon

In the boxes above is the triplet value (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field.

Element type 3 corresponds to a polygon. A polygon has implied area. Post Oracle 8.1.5, instead of an element type 3, outer ring polygons are represented as element type 1003, and inner ring (void) polygons are represented as element type 2003. Outer ring polygons must be stored with a counter-clockwise 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 with the Open GIS Consortium (OGC) standard.

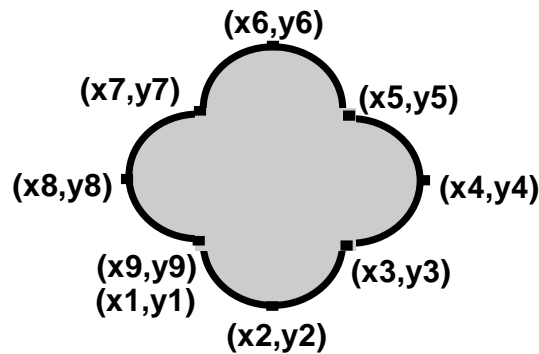
The interpretation field will define how to connect the segments that make up the polygon. In this example, the interpretation is 1, and corresponds to a polygon whose segments are connected with straight lines (a conventional polygon).

Line segments of polygons cannot cross each other.

Note: The SDO_GTYPE for this geometry is 2003 (single polygon element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Element Example: Arc Polygon

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1003 | 2 |



- **Interpretation 2: All line segments are circular arcs**
- **Area is implied**
- **Arcs cannot cross each other**
- **Order (rotation) matters**

ORACLE

3-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Arc Polygon

In the boxes above is the triplet value (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field.

Element type 3 corresponds to a polygon. A polygon has implied area. Post Oracle 8.1.5, instead of an element type 3, outer ring polygons are represented as element type 1003, and inner ring (void) polygons are represented as element type 2003. Outer ring polygons must be stored with a counter clockwise 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 with the Open GIS Consortium (OGC) standard.

The interpretation field will define how to connect the segments that make up the polygon. In this example, the interpretation is 2, and 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.

Line segments of polygons cannot cross each other.

Note: The SDO_GTYPE for this geometry is 2003 (single polygon element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Element Example: Rectangle

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1003 | 3 |



- **Optimal storage:** Defined by lower left point, upper right point
- **Area is implied**

ORACLE

3-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Rectangle

In the boxes above is the triplet value (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field.

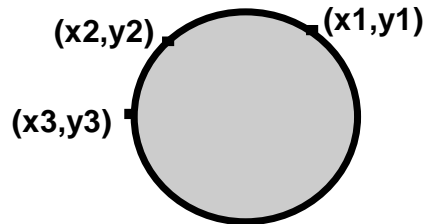
Element type 3 corresponds to a polygon. A polygon has implied area. Post Oracle 8.1.5, instead of an element type 3, outer ring polygons are represented as element type 1003, and inner ring (void) polygons are represented as element type 2003.

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

Note: The SDO_GTYPE for this geometry is 2003 (single polygon element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Element Example: Circle

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1003 | 4 |



- Defined by any three distinct points on the circumference
- Area is implied
- Order (rotation) matters

ORACLE

3-26

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Circle

In the boxes above is the triplet value (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field.

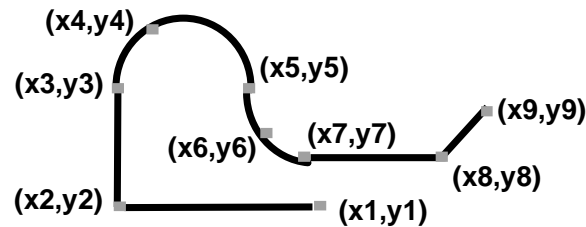
Element type 3 corresponds to a polygon. A polygon has implied area. Post Oracle 8.1.5, instead of an element type 3, outer ring polygons are represented as element type 1003, and inner ring (void) polygons are represented as element type 2003. Outer ring polygons must be stored with a counter clockwise rotation. Inner ring (void) polygons must be stored with a clockwise rotation. Enforcing the rotation enables faster area calculations.

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

Note: The SDO_GTYPE for this geometry is 2003 (single polygon element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Element Example: Compound Line String

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 4 | 3 |
| 1 | 2 | 1 |
| 5 | 2 | 2 |
| 13 | 2 | 1 |



- First triplet (header) defines the number of subelements
- Subelements must be contiguous
- Arcs and line segments of subelements can cross
- Element type 4 can *only* contain subelements of type 2

ORACLE

3-27

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Compound Line String

In the boxes above are the triplet values (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field.

Element type 4 corresponds to a compound line string. Line strings discussed earlier (element type 2) were either all connected with straight lines or all connected with circular arcs. Think of compound as combination. With a compound line string, you can define a single contiguous line string 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 will be 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.

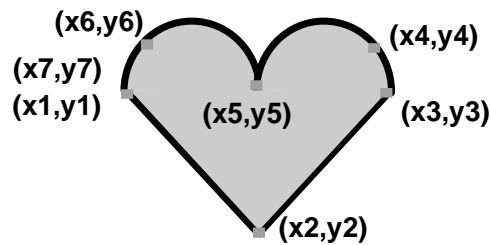
All the subelements must be of element type 2 (interpretation 1 or 2), and must be contiguous (that is, the last point of a subelement is the first point of the next subelement).

Notice in the example above, the ordinate offsets of each subelement are different, and correspond to where the subelement begins in the ordinate array.

Note: The SDO_GTYPE for this geometry is 2002 (single line string element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Element Example: Compound Polygon

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1005 | 2 |
| 1 | 2 | 1 |
| 5 | 2 | 2 |



- First triplet (header) defines the number of subelements
- Subelements must be contiguous
- Arcs and line segments of subelements cannot cross
- Element type 5 can *only* contain subelements of type 2
- Area is implied

ORACLE

3-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Compound Polygon

In the boxes above are the triplet values (ordinate offset, element type, interpretation) you would store in the SDO_ELEM_INFO field.

Element type 5 corresponds to a compound polygon. Polygons discussed earlier (element type 3) were either all connected with straight lines or all connected with circular arcs. Think of compound as combination. With a compound polygon, you can define a single polygon that is made up of contiguous straight lines and circular arcs.

Compound polygons were not added to make the product more complex. A compound polygon is often the result of a buffer operation which we will discuss in a future lesson. If you buffer a line, the result is a compound polygon (that is, straight lines on either side of the line you buffered and half circles around the end points).

Compound polygons have the same properties as polygons (that is, cannot self-cross, and so on). Post Oracle 8.1.5, instead of an element type 5, outer ring compound polygons are represented as element type 1005, and inner ring (void) compound polygons are represented as element type 2005. Outer ring compound polygons must be stored with a counter clockwise rotation. Inner ring (void) compound polygons must be stored with a clockwise rotation. Enforcing the rotation enables faster area calculations.

Element Example: Compound Polygon (continued)

A compound polygon is defined by a series of contiguous subelements. Each subelement will be 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 must be of 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).

Notice in the example above, the ordinate offsets of each subelement are different, and correspond to where the subelement begins in the ordinate array.

Note: The SDO_GTYPE for this geometry is 2003 (single polygon element). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (that is, the big picture).

Details of Polygon Element Types

| Post 8.1.5 Element Types | Interpretation |
|-------------------------------------|---|
| 1003 OUTER_RING_POLYGON | 1, 2, 3 or 4 |
| 2003 INNER_RING_POLYGON | 1, 2, 3 or 4 |
| 1005 OUTER_RING COMPOUND_POLYGON | # of type 2 elements that make up the polygon |
| 2005 INNER_RING COMPOUND_POLYGON | # of type 2 elements that make up the polygon |

- **Optimized element types for polygons with voids**
- **Outer rings (1003, 1005) must be followed by all its inner rings (2003, 2005)**
- **Outer rings (1003, 1005) must have counterclockwise orientation**
- **Inner rings (2003, 2005) must have clockwise orientation**
- **8.1.6 includes a tool to migrate data to the new element types**

ORACLE

3-30

Copyright © Oracle Corporation, 2001. All rights reserved.

Details of Polygon Element Types

This slide summarizes the rules for representing polygons.

Post Oracle 8.1.5, instead of an element type 3, outer ring polygons are represented as element type 1003, and inner ring (void) polygons are represented as element type 2003.

Post Oracle 8.1.5, instead of an element type 5, outer ring compound polygons are represented as element type 1005, and inner ring (void) compound polygons are represented as element type 2005.

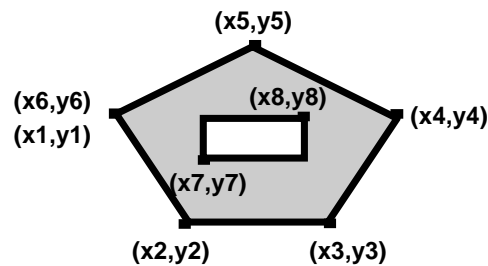
Outer ring compound polygons must be stored with a counter clockwise rotation. Inner ring (void) compound polygons must be stored with a clockwise rotation. Enforcing the rotation enables faster area calculations.

Post Oracle 8.1.5, if using the four digit element types for polygons (or compound polygons), an outer ring polygon must be followed by all its inner rings (voids). When reading geometries, this makes it easier to determine which rings are voids and which are not. Prior to this order enforcement, it was up to the application to mathematically determine which rings were outer rings and which were voids.

A migration tool, SDO_MIGRATE.TO_CURRENT takes 8.1.5 formatted polygons (or compound polygons) and converts them to polygons that conform to the rules described above.

Element Example: Polygon with a Void

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1003 | 1 |
| 13 | 2003 | 3 |



- A void can be modeled with an element of type 2003 or 2005
- Voids can contain islands, and islands can contain voids
- Area is computed as the difference between the outer and inner polygons
- Order (rotation) matters

ORACLE

3-31

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: 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.

Area is computed as the difference between outer and inner polygons.

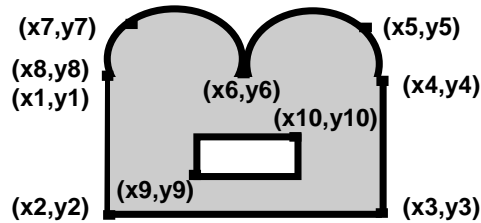
Rotation of outer rings elements must be counter clockwise, and inner ring elements must be clockwise.

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: The SDO_GTYPE of this geometry is 2003 (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).

Element Example: Compound Polygon with a Void

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1005 | 2 |
| 1 | 2 | 1 |
| 7 | 2 | 2 |
| 17 | 2003 | 3 |



- A void can be modeled with any combination of type 2003 and type 2005 elements
- Voids can contain islands, and islands can contain voids
- Area is implied as the difference between the outer and inner polygons
- Order (rotation) matters

ORACLE

3-32

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Compound Polygon with a Void

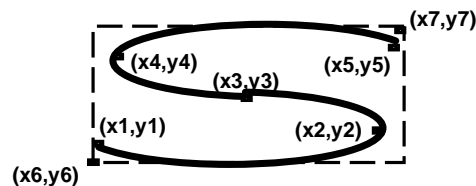
This is an example of a compound polygon with a void. The outer ring has an element type of 1005 (compound polygon), and the inner ring has an element type of 2003 (optimized rectangle).

The compound polygon has 3 triplet value entries in the SDO_ELEM_INFO field. The first triplet is a header (1,1005,2). The interpretation of 2 denotes two more triplet values follow to define the compound polygon subelements (1,2,1) and (7,2,2). Notice 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: The SDO_GTYPE of this geometry is 2003 (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).

Element Example: Unknown Element

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 0 | 1324 |
| 11 | 1003 | 3 |



- Element type 0 is for modeling unsupported element types (for example, curves, splines, and so on)
- Element type 0 is ignored by Oracle Spatial
- A geometry with an element type 0 must contain at least one element of type 1,2,3,4 or 5. The non-0 element is an approximation of the unsupported element. The approximation is indexed by Oracle Spatial.

ORACLE

3-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Example: Unknown Element

At this point, you know all of the supported element types. If you want to model something that Oracle Spatial does not natively support (for instance, a curve or a spline), you can use an element type 0 (unknown element).

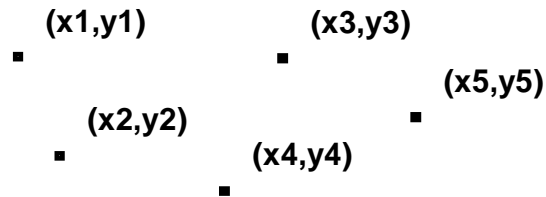
A geometry with an element type 0 must contain at least one element of type 1, 2, 1003, 2003, 4, 1005, or 2005. The non-0 element is an approximation of the unsupported element. The approximation is indexed and queried by Oracle Spatial.

Note: Since elements of type 0 contain customized information about an unsupported element type, applications that generically read/write Oracle Spatial may not be able to read/write geometries that contain elements of type 0.

Note: The SDO_GTYPE of this geometry is 2003 (single 2D polygon). The SDO_GTYPE of a geometry is determined by the supported element types (that is, the rectangle element in the example above).

Collection Example: Multipoint

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1 | 5 |



- Interpretation is the number of points in the collection

ORACLE

3-34

Copyright © Oracle Corporation, 2001. All rights reserved.

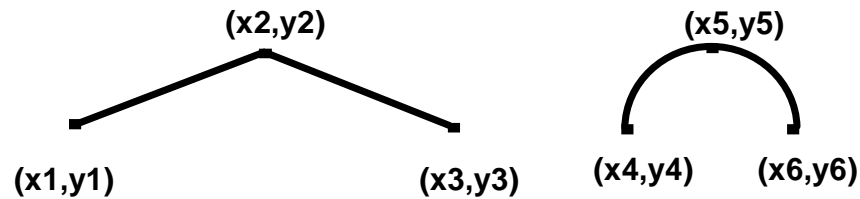
Collection Example: Point Cluster

This is the first example of a collection geometry. A collection is a geometry that contains more than one element. 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: The SDO_GTYPE for this geometry is 2005 (2D multipoint). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Collection Example: Multiline String

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 2 | 1 |
| 7 | 2 | 2 |



- Elements can be line strings, circular arc strings, or compound line strings

ORACLE

3-35

Copyright © Oracle Corporation, 2001. All rights reserved.

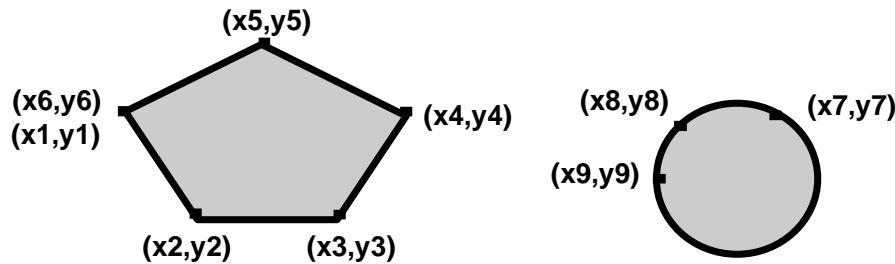
Collection Example: Multiline String

A collection is a geometry that contains more than one element. 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 above, the straight line string triplet is (1,2,1) and the circular arc string triplet is (7,2,2).

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

Collection Example: Multipolygon

| Ordinate offset | Element type | Interpretation |
|-----------------|--------------|----------------|
| 1 | 1003 | 1 |
| 13 | 1003 | 4 |



- Elements can be polygons, arc polygons, or compound polygons

ORACLE

3-36

Copyright © Oracle Corporation, 2001. All rights reserved.

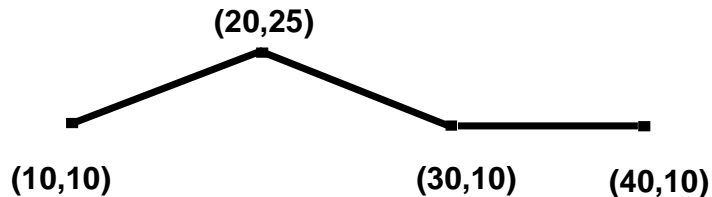
Collection Example: Multipolygon

A collection is a geometry that contains more than one element. 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 above, the pentagon polygon element triplet is (1,1003,1) and the circle triplet is (13,1003,4).

Note: The SDO_GTYPE for this geometry is 2007 (2D multipolygon). Remember, the SDO_GTYPE is the first field of the SDO_GEOMETRY object, and describes the entire geometry (the big picture).

Constructing Geometries

```
SQL> INSERT INTO LINES VALUES (  
2>     attribute_1, ... attribute_n,  
3>     MDSYS.SDO_GEOMETRY (  
4>         2002, null, null,  
5>         MDSYS.SDO_ELEM_INFO_ARRAY (1,2,1),  
6>         MDSYS.SDO_ORDINATE_ARRAY (  
7>             10,10, 20,25, 30,10, 40,10))  
8> );
```



ORACLE

3-37

Copyright © Oracle Corporation, 2001. All rights reserved.

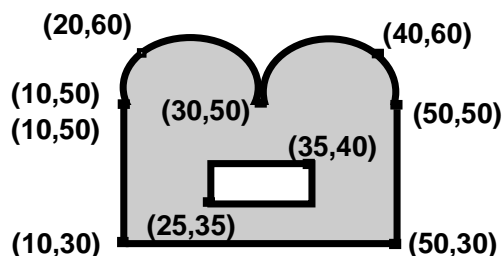
Constructing Geometries

Earlier, you saw a transactional insert for a point 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 SDO_ELEM_INFO_ARRAY and SDO_ORDINATE_ARRAY constructors to populate the array fields.

Constructing Geometries

```
SQL> INSERT INTO PARKS VALUES(  
2>   attribute_1, ..., attribute_n,  
3>   MDSYS.SDO_GEOMETRY(  
4>     2003, null, null,  
5>     MDSYS.SDO_ELEM_INFO_ARRAY  
6>       (1,1005,2, 1,2,1, 7,2,2, 17,2003,3),  
7>     MDSYS.SDO_ORDINATE_ARRAY  
8>       (10,50,10,30,50,30,50,50,40,60,  
9>        30,50,20,60,10,50,25,35,35,40)));
```



ORACLE

3-38

Copyright © Oracle Corporation, 2001. All rights reserved.

Constructing Geometries

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 is inserting the compound polygons with a void example 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: The SDO_GTYPE of this geometry is 2003 (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).

Element Types Summarized

| <u>Number</u> | <u>Element Type</u> | <u>Interpretation</u> |
|---------------|---------------------|---|
| 0 | UNKNOWN_ELEMENT | |
| 1 | POINT | # of points in collection |
| 2 | LINESTRING | 1 - Straight lines 2 - Circular arcs |
| 3 | POLYGON | 1 - Straight lines |
| 1003 | (Outer) | 2 - Circular arcs |
| 2003 | (Inner) | 3 - Optimized rectangle |
| | | 4 - Circle |
| 4 | COMPOUND LINESTRING | # of line string subelements that make up the line string |
| 5 | COMPOUND POLYGON | # of line string subelements that make up the polygon |
| 1005 | (Outer) | |
| 2005 | (Inner) | |

ORACLE

3-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Element Types Summarized

This slide should make more sense now. It summarizes how to represent an element in the SDO_ELEM_INFO field of the SDO_GEOMETRY object.

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 will have a triplet value entry in the SDO_ELEM_INFO array (ordinate offset, element type, interpretation).

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

Do not confuse element types with SDO_GTYPEs. SDO_GTYPEs are the big picture. SDO_GTYPEs represent what type of geometry is stored. A geometry is made up of one or more elements, and the element type describes the element.

In Oracle 8.1.5, the element types were all one digit (0-5). Post 8.1.5, we introduced a 4 digit element type for polygon elements. Element type 3 is replaced with either 1003 (outer ring) or 2003 (void), and element type 5 is replaced with either 1005 (outer ring) or 2005 (void). The 4 digit element type for polygons was introduced to make it easier to determine if a polygon element is an outer ring or a void.

Element Types Summarized (continued)

Element type 0: To model an element that is not currently supported by Oracle Spatial.

Element type 1: For storing multipoints. The interpretation field describes how many points are in the multipoint.

Element type 2: For storing line strings. Interpretation 1 represents a conventional line string connected with straight lines. Interpretation 2 represents an arc line string made up of one or more circular arcs.

Element type 3: Post 8.1.5, 1003 represents an outer ring polygon, 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, only requires 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 area)

Element type 4: Compound line string, 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: Post 8.1.5, 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.

Syntax for Flattening the VARRAYs

```
SELECT *  
FROM TABLE (SELECT a.geom.sdo_ordinates  
                FROM geod_states a  
                WHERE a.state = 'California');
```

```
SELECT *  
FROM TABLE (SELECT a.geom.sdo_elem_info  
                FROM geod_states a  
                WHERE a.state = 'California');
```

Can only flatten one VARRAY at a time

ORACLE

3-41

Copyright © Oracle Corporation, 2001. All rights reserved.

Syntax for Flattening the VARRAYs

This first query returns the list of SDO_ORDINATES for a single geometry. The second query returns the list of SDO_ELEM_INFO entries for a single geometry.

TABLE is a keyword in Oracle.

Spatial Metadata

- The spatial routines require you to populate a view that contains metadata about SDO_GEOMETRY columns
- The metadata view is created for all Oracle Spatial users when Oracle Spatial is installed
- The metadata view is called USER_SDO_GEOM_METADATA
- For every SDO_GEOMETRY column, insert a row in the USER_SDO_GEOM_METADATA view

ORACLE

3-42

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

The information in this view is used by Oracle Spatial and by third party tools and applications. The information in this view is only maintained outside of Oracle Spatial. Oracle Spatial will never update or delete information in the USER_SDO_GEOM_METADATA view.

USER_SDO_GEOM_METADATA

```
SQL> DESCRIBE USER_SDO_GEOM_METADATA
```

| Name | Null? | Type |
|-------------|----------|---------------------|
| ----- | ----- | ----- |
| TABLE_NAME | NOT NULL | VARCHAR2(32) |
| COLUMN_NAME | NOT NULL | VARCHAR2(32) |
| DIMINFO | | MDSYS.SDO_DIM_ARRAY |
| SRID | | NUMBER |

- **MDSYS.SDO_DIM_ARRAY**

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

- **MDSYS.SDO_DIM_ELEMENT object**

| | |
|---------------|--------------|
| SDO_DIMNAME | VARCHAR2(64) |
| SDO_LB | NUMBER |
| SDO_UB | NUMBER |
| SDO_TOLERANCE | NUMBER |

ORACLE

3-43

Copyright © Oracle Corporation, 2001. All rights reserved.

USER_SDO_GEOM_METADATA

This slide shows you 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 (varying sized array) of SDO_DIM_ELEMENT objects. This is where you define the axis for your coordinate system (2, 3, or 4 axis can be defined). The fields of the SDO_DIM_ELEMENT object are described on the next slide.

SRID: SRID is an acronym for spatial reference system id (SRID). 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. Coordinate systems will be discussed in more detail in the “Coordinate Systems Overview” lesson.

USER_SDO_GEOM_METADATA SDO_DIM_ELEMENT Object

- **SDO_DIMNAME**
 - The dimension name
- **SDO_LB**
 - Lowest possible value for this dimension
- **SDO_UB**
 - Largest possible value for this dimension
- **SDO_TOLERANCE**
 - Distance required for two numbers not to be considered the same number. Used by Oracle Spatial indexing, operators and functions
 - Geodetic data: Data stored in longitude/latitude, tolerance value is in meters (that is, 0.5 is a one-half meter tolerance)
 - Projected data: Tolerance is the same units as the data. Match decimal precision of ordinates with zeros and append a 5 (for example, 0.0000005)

ORACLE

3-44

Copyright © Oracle Corporation, 2001. All rights reserved.

USER_SDO_GEOM_METADATA: SDO_DIM_ELEMENT Object

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

SDO_DIMNAME: A character string to identify this 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.

For quadtree indexes, it is critical that all geometry ordinates fall in the range of SDO_LB to SDO_UB for each dimension, and recommended to have the same upper and lower bounds for all spatial data in a project. R-tree indexes are more flexible, and can accommodate geometries that fall outside of the SDO_LB to SDO_UB range. For geodetic data, the bounds of the coordinate system should always be set to -180,180 and -90,90.

USER_SDO_GEOM_METADATA: SDO_DIM_ELEMENT Object (continued)

SDO_TOLERANCE: The distance two numbers must be from each other not to be considered the same number. For Geodetic data (that is, data represented in longitude/latitude), this value is specified in meters (that is, 0.005 is a 5 millimeter tolerance). For projected data (that is, not longitude/latitude data, or no SRID set) this value is in the same units as the data. Match the decimal precision of ordinates with zeros and append a 5 (that is, 0.000000005).

Note: In Oracle9i and beyond, for geodetic data, try not to use a smaller tolerance value than is necessary, because this causes additional precision requirements for all spatial operations, which may result in slower performance.

Note: Before Oracle9i, for geodetic data, match the decimal precision of ordinates with zeros and append a 5 (that is, 0.000000005).

Populating the USER_SDO_GEOM_METADATA View

```
SQL> INSERT INTO USER_SDO_GEOM_METADATA
2> (TABLE_NAME, COLUMN_NAME, DIMINFO, SRID)
3> VALUES (
4>   'ROADS',
5>   'GEOMETRY',
6>   MDSYS.SDO_DIM_ARRAY (
7>     MDSYS.SDO_DIM_ELEMENT('Long', -180, 180, 0.5),
8>     MDSYS.SDO_DIM_ELEMENT('Lat', -90, 90, 0.5)),
9>   8307);
```

ORACLE

3-46

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

Notes on Dimensions

- All layers that you want to compare *must* have the *exact same* axis bounds, if they have the same SRID (quadtree index only)!
- You can place more than two dimensions in the SDO_DIM_ARRAY
- Quadtree index is built on first two dimensions
- R-tree index is built on up to four dimensions
- Primary filter is two dimensional for quadtree
- Primary filter is up to four dimensional for R-tree
- Secondary filters are always 2D (index type does not matter)

ORACLE

3-47

Copyright © Oracle Corporation, 2001. All rights reserved.

Notes on Dimensions

If you are using quadtree indexes, all the layers you want to compare must have the exact same axis bounds if they have the same SRID. By compare, we mean using a geometry out of one layer as an area of interest against another layer.

With R-tree indexes, the axis bounds across layers with the same SRID do not need to match.

You can place more than two dimensions in the SDO_DIM_ARRAY. One scenario might be for LRS geometries (that is, storing measure values with each point). You create a spatial index and not include the measure values. Another scenario is to store Z values (or heights) with each point.

Quadtree spatial indexes only index the first two dimensions of each point. All other dimensions are skipped. Primary filter is always two dimensional.

Notes on Dimensions (continued)

R-tree spatial indexes can index up to four dimensions, and can perform up to four dimensional primary filter queries.

If you create an R-tree index on more than two dimensions, the only supported spatial operator is `SDO_FILTER`.

Regardless of which index type you choose, R-tree or quadtree, secondary filters are always two dimensional. Volumetric (that is, 3D) secondary filters are not currently provided. If you have a volumetric secondary filter algorithm, you can implement it in a stored procedure (that is, Java or PL/SQL), and call your 3D secondary filter on the results of Oracle's 3D primary filter (all in the same SQL statement).

*** _SDO_GEOM_METADATA System Views**

There are three * _SDO_GEOM_METADATA system views:

- **USER_SDO_GEOM_METADATA**
 - Insert into this view
- **ALL_SDO_GEOM_METADATA**
 - Select from this view
- **DBA_SDO_GEOM_METADATA**
 - Can see all spatial metadata

ORACLE

3-49

Copyright © Oracle Corporation, 2001. All rights reserved.

*** _SDO_GEOM_METADATA System Views**

There are three * _SDO_GEOM_METADATA system views. These are created for you during installation.

USER_SDO_GEOM_METADATA: This is the view you insert into. For every SDO_GEOMETRY column, insert one row of metadata into the USER_SDO_GEOM_METADATA view.

ALL_SDO_GEOM_METADATA: Only select from this system view. Never insert directly into this view. This view will contain all the metadata for the Oracle user you are logged in as, and all metadata for geometry columns that user has select access to.

DBA_SDO_GEOM_METADATA: Only select from this system view. Never insert directly into this view. This view is available to DBAs, and will contain the metadata for all the geometry columns in the database.

Summary

In this lesson, you should have learned how to:

- **Describe the schema associated with a spatial layer**
- **Describe how spatial data is stored using the Oracle Spatial object-relational model**
- **Create a table with a spatial layer**
- **Load the metadata view associated with a spatial layer**

ORACLE

4

Coordinate Systems Overview

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Understand coordinate systems**
- **Understand the difference between georeferenced and non-georeferenced data**
- **Understand a geodetic coordinate system**
- **Understand a projected coordinate system**
- **Understand how to associate a coordinate system with a spatial layer**
- **Understand how to add coordinate system information to an existing spatial layer**

ORACLE

4-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

This section is intended to introduce you to the concepts associated with coordinate systems. This lesson is an overview only.

A future chapter will include detailed information about coordinate systems and how they are used in Oracle Spatial.

Coordinate Systems Concepts

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

ORACLE

4-3

Copyright © Oracle Corporation, 2001. All rights reserved.

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 are relative to each other within the specified coordinate system.

Whenever dealing with spatial data, there is a coordinate system associated with that data.

In Oracle Spatial, the coordinate system can be implicitly defined, but not specified (that is, 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 USER_SDO_GEOM_METADATA and the SDO_SRID field for each geometry in the layer.

Coordinate Systems Concepts

- **Two types of coordinate systems**
 - **Georeferenced (related to a specific representation of Earth)**
 - **Local, or non-georeferenced (not related to any representation of Earth)**
- **Georeferenced can be either of two types**
 - **Geodetic (longitude/latitude)**
 - **Projected**

ORACLE

4-4

Copyright © Oracle Corporation, 2001. All rights reserved.

More Coordinate Systems Concepts

In Oracle Spatial at a high level there are two kinds of coordinate systems. Georeferenced coordinate systems are tied to a specific representation of the Earth. Local, or non-georeferenced coordinate systems have no relationship to any representation of the Earth.

Oracle Spatial supports two kinds of georeferenced data:

- **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 which vary 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 length of parallels decreases as the parallels move away from the equator until their length equals 0 at the North and South Poles. 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**, 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.

Projected CS Concepts

- **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

4-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Projected Data

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 will define one and only one closed area.

Geodetic CS 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 connects those two points using the great circle.

Consider the two points (180,85) and (0,85)

- The shortest path connecting these two points does not follow the 85 degree latitude line.
- It goes through the North Pole!

ORACLE

4-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Geodetic Data

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

A great circle is the largest circle that can be drawn on the Earth's surface and the shortest distance between any two places on Earth follow a great circle route.

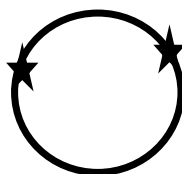
If the Earth was a perfectly symmetrical geometry, the great circle would always divide the Earth exactly in half.

Since the connection between the points is made through a circle, there are two different ways to connect any two points using the great circle.

The great circle used to connect points doesn't necessarily follow longitude/latitude lines.

Geodetic CS Concepts

- A great circle goes through any two given points on the Earth's surface.
- The great circle can connect any two points on the Earth in two different ways, and represent two pieces of the circle.
 - Oracle Spatial always takes the smaller piece as the intended line.
 - A line defined by two points must be less than half the distance defined by the great circle.



ORACLE

4-7

Copyright © Oracle Corporation, 2001. All rights reserved.

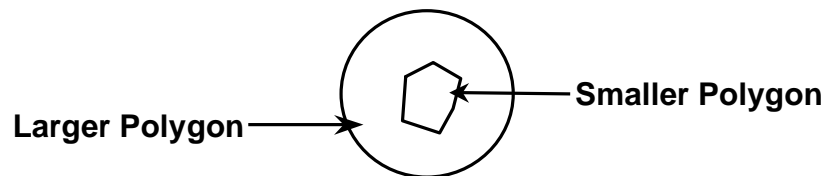
The Great Circle

Given that a great circle can be constructed that connects any two points on the Earth's surface, and the minimum distance between any two points on the surface of the Earth is always defined via the smaller of the two parts of the great circle:

- Oracle Spatial always uses the great circle to connect two points, and chooses the minimum distance between those points as the line defined by the two points.
- Two points in Oracle Spatial must always define a distance less than 1/2 the circumference of the Earth. Oracle Spatial always chooses the shortest distance between two points. If the length of a line is exactly 1/2 the circumference of the Earth, Oracle Spatial currently has no mechanism of determining which line is the appropriate line.

Geodetic CS Concepts

- A polygon can define two closed areas.
- Oracle Spatial takes the smaller polygon as the intended polygon.
 - A polygon area is required to be less than half the surface area of the Earth.



ORACLE

4-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Polygons in a Geodetic Coordinate System

When using a geodetic coordinate system, any polygon defines two closed areas.

In the example above there are two polygons. The smaller area specified by that polygon is light colored. The larger area specified by the same polygon is dark colored, and the area wraps completely around the Earth. Oracle Spatial will always use the smaller of the two polygons (the lighter polygon in this case).

Any single polygon in Oracle Spatial must be less than half the surface area of the Earth. There is currently no way to describe a polygon that is equal to or greater than half the surface area of the Earth within Oracle Spatial. This restriction may be removed in a future release.

Geodetic Geometry Examples

- Is this a valid polygon?

```
MDSYS.SDO_GEOMETRY(2003, 8307, NULL,  
  MDSYS.SDO_ELEM_INFO_ARRAY(1, 1003, 1),  
  MDSYS.SDO_ORDINATE_ARRAY(180, 60,  
                              -110, 60,  
                              0, 60,  
                              110, 60,  
                              180, 60))
```

ORACLE

4-9

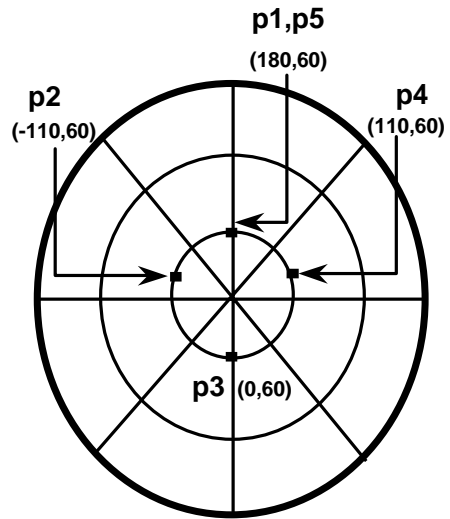
Copyright © Oracle Corporation, 2001. All rights reserved.

An Interesting Polygon

In projected space, the above polygon would not be a legal polygon. It would look like a straight line. In geodetic space, the polygon specified would be a surface that covers the northern hemisphere starting at 60 degrees north latitude.

Geodetic Geometry Examples

| | | | |
|-----------|--------|----------|----------|
| p2 | p3 | p4 | p1,p5 |
| (-110,60) | (0,60) | (110,60) | (180,60) |
| | | | |



ORACLE

4-10

Copyright © Oracle Corporation, 2001. All rights reserved.

An Interesting Polygon (continued)

Above, you can see the difference between the previous polygon geometry specified in projected space where it is not a valid polygon and in geodetic space, where it is. In geodetic space looking down over the North Pole, the polygon described by the geometry is the center portion of the illustrated geometry.

Coordinate Systems in Oracle Spatial

- Over 950 different predefined coordinate systems provided
- Definitions kept in table **MDSYS.CS_SRS**

```
describe mdsys.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 |

- Coded using the notation defined by the Open GIS Consortium
- Each coordinate system identified by unique number (SRID)

ORACLE

4-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle Spatial and Coordinate Systems

Oracle Spatial provides over 950 predefined coordinate systems.

All of the coordinate systems are defined in an Oracle dictionary table owned by user MDSYS. The table name is CS_SRS.

Information about each coordinate system is encoded using the notation defined by the Open GIS Consortium (OGC). This information is stored in the WKTEXT column of the CS_SRS table.

Each coordinate system in Oracle Spatial is assigned a unique number, which is stored in the SRID column.

The MDSYS Oracle user has granted select access to CS_SRS to public.

MDSYS.CS_SRS Table

- The SRID column contains a unique Spatial Reference ID number.
- The WKTEXT column contains the well-known text description.
 - Geodetic SRIDs when WKTEXT column begins with 'GEOGCS'
 - Projected SRIDs when WKTEXT column begins with 'PROJCS'
- The CS_NAME field contains a short name for the coordinate system.
- The AUTH_NAME field currently contains 'Oracle'.
- The AUTH_SRID field currently contains the SRID.
- The CS_BOUNDS field is currently NULL.

ORACLE

4-12

Copyright © Oracle Corporation, 2001. All rights reserved.

The MDSYS.CS_SRS Table

There are six fields in the CS_SRS table:

- The SRID field contains the unique Spatial Reference ID number. The Spatial Reference System is the same as the coordinate system.
- The WKTEXT column holds the well-known text as described by the Open GIS Consortium.
- 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, the AUTH_NAME will always be 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 is set aside for Oracle use and is currently NULL. It eventually will contain a geometry specified in WGS-84 (longitude/latitude) which is the valid polygon boundary for each projection. It will remain NULL for geodetic data.

Associating Geometries with Coordinate Systems

- **Defined for all geometries in a layer**
 - In the SRID column of the USER_SDO_GEOM_METADATA view
- **Defined for individual geometries**
 - In the SDO_SRID field of the SDO_GEOMETRY object
 - Same SRID for all geometries in a layer
- **Must be correctly defined and the same in both, or NULL in both**

ORACLE

4-13

Copyright © Oracle Corporation, 2001. All rights reserved.

Associating Geometries with Coordinate Systems

There are two entries required to associate geometries with a Spatial Reference System.

- There is one entry defined for all geometries in the entire layer in the SRID field of the USER_SDO_GEOM_METADATA view.
- Each individual geometry in a layer should have the SDO_SRID field within the SDO_GEOMETRY object set. Every geometry in the layer should have the same SRID value in the SDO_SRID field.

The value of the SRID field in the USER_SDO_GEOM_METADATA view must match the values in the SDO_SRID field of each geometry in the layer. The SRID and SDO_SRID fields can both be NULL, or they can be set to a valid value. The valid values can be found in the SRID column of the MDSYS.CS_SRS table. The next slide shows some valid values for SDO_SRID.

Adding Coordinate Systems Information to Existing Layers

To add coordinate systems information to existing layers:

- **Drop the spatial index if it exists**
- **Add SRID to USER_SDO_GEOM_METADATA**
- **Add SDO_SRID to each geometry in the layer**
- **Adjust TOLERANCE and/or coordinate system bounds if required (almost always required for geodetic data)**
- **Rebuild the spatial index**

ORACLE

4-14

Copyright © Oracle Corporation, 2001. All rights reserved.

Adding Coordinate Systems Information to Existing Layers

With the new whole Earth geometry model in Oracle9i Spatial, the increase in functionality makes it much more attractive to include coordinate systems information in geodetic spatial layers. Adding the information for a coordinate system to a layer may require several steps.

An overview of the process is to do the following:

- If a spatial index exists on the layer, it must be dropped
- Add a value for the SRID field associated with the layer in the USER_SDO_GEOM_METADATA view
- Add a value for SDO_SRID to every geometry in the layer
- Update the values for TOLERANCE and/or adjust the coordinate system bounds for the layer if required
- If there was a spatial index on the layer, rebuild it

Adding Coordinate Systems Information Example

- Existing STATES table includes a geodetic layer with no associated coordinate systems
- Drop the spatial index if it exists

```
SQL> DROP INDEX STATES_SIDX;
```

- Add SRID to USER_SDO_GEOM_METADATA

```
SQL> UPDATE USER_SDO_GEOM_METADATA  
      SET SRID=8307  
      WHERE TABLE_NAME='STATES'  
      AND COLUMN_NAME='GEOM' ;
```

- Add SDO_SRID to each geometry in the layer.

```
SQL> UPDATE STATES A SET A.GEOM.SDO_SRID=8307;
```

ORACLE

Adding Coordinate Systems Information to Existing Layers Example

The process of adding coordinate systems information to a layer starts by dropping the spatial index on the layer if it exists.

After ensuring there is no spatial index on the data, it is important to associate a spatial reference system id (SRID) with the layer and with each geometry.

The layer level operation is done by updating the SRID field of the USER_SDO_GEOM_METADATA view entry for the table/column associated with the layer.

An SDO_SRID value is added to each geometry in the layer by using an update statement. Note that to manipulate a field in an object column (in this case, updating the SDO_SRID field of the SDO_GEOMETRY object) a table alias must be used. The table alias in this example is A. Also note that this operation that updates every single SDO_GEOMETRY object loaded for the layer.

Adding Coordinate Systems Information Example

- Adjust TOLERANCE and/or coordinate system bounds if required (almost always required for geodetic data).

```
SQL> UPDATE USER_SDO_GEOM_METADATA A
      SET A.DIMINFO =
          MDSYS.SDO_DIM_ARRAY(
              MDSYS.SDO_DIM_ELEMENT('Longitude',-180,180,0.5),
              MDSYS.SDO_DIM_ELEMENT('Latitude',-90,90,0.5))
      WHERE TABLE_NAME = 'STATES' and COLUMN_NAME = 'GEOM';
```

- Rebuild the spatial index.

```
SQL> CREATE INDEX STATES_SIDX ON STATES(GEOM)
      INDEX_TYPE IS MDSYS.SPATIAL_INDEX;
```

ORACLE

4-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Adding Coordinate Systems Information to Existing Layers Example (continued)

In order to get the complete benefits of a geodetic coordinate system, Oracle Spatial requires that the coordinate system bounds be set to -180 to 180 (Longitude) and -90 to 90 (Latitude). Additionally, the tolerance value must be specified in meters for geodetic coordinate systems.

After all of the changes have been made to add the appropriate coordinate systems information to the layer metadata as well as the geometry objects themselves, the spatial index can be rebuilt. More information about indexes will be included in the next lessons.

Common Coordinate Systems

- **Some common coordinate systems include:**
- **Geodetic**
 - NAD 83 (SRID=8265)
 - WGS 84 (SRID=8307)
- **Projected**
 - **Continental/country specific/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

4-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Common Coordinate Systems

Some of the most common coordinate systems used for geodetic data are:

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

There are a large number of projections in use throughout the world. The most accurate projections describe smaller areas. In the United States, there are a number of state plane projections. Smaller states might require only one projection for a given unit (such as meter or foot), and larger states will require more projections to cover the state with the same amount of precision. Typically, the larger the projection the less accurate it will be.

Restrictions on Geodetic Coordinate Systems

- **Element types not supported**
 - Circular arcs or elements that include arcs (compound elements)
 - Circles
 - Optimized rectangles
- **Element type restrictions**
 - No single line segment equal to or greater than 1/2 the length of the Earth's great circle (can break into multiple segments)
 - No polygon with area equal to or greater than 1/2 area of the Earth
- **Features not supported**
 - SDO_GEOM.SDO_MBR and associated functions

ORACLE

4-18

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions with Geodetic Coordinate Systems

There are some restrictions when using geodetic coordinate systems.

With regards to geometric primitive types, restrictions include:

- A geometry cannot contain any arcs. This means that line strings composed of circular arcs are not supported (element type 2, interpretation 2), and compound line strings that have subelements that are circular arcs are not supported. Additionally, polygons that contain one or more arcs are not supported.
- Optimized circles (element type 1003 or 2003, interpretation 4) are not supported.
- The optimized rectangle type (element type 1003 or 2003, interpretation 3) is not supported.

Also, there are restrictions on the size of certain elements.

- A line segment within a line string cannot be equal to or greater than 1/2 the length of the Earth's great circle. A line segment is the portion of a line defined by two coordinates.
- Also, the area of a polygon cannot be equal to or greater than 1/2 the area of the Earth. This restriction may change in the future.

SDO_GEOM.SDO_MBR and the associated functions

SDO_GEOM.SDO_MAX_MBR_ORDINATE ,

SDO_GEOM.SDO_MIN_MBR_ORDINATE, and SDO_AGGR_MBR are not supported with geodetic coordinate systems.

Summary

In this lesson, you should have learned how to:

- **Define coordinate systems**
- **Explain the difference between georeferenced and non-georeferenced coordinate systems**
- **Define a geodetic coordinate system**
- **Define a projected coordinate system**
- **Associate a coordinate system with a spatial layer**
- **Add coordinate system information to an existing spatial layer**

ORACLE

4-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

This section was intended to be an introduction to the concepts associated with coordinate systems. This lesson was an overview only.

A future chapter will include detailed information about coordinate systems and how they are used in Oracle9i Spatial.

5

Loading Spatial Data

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Identify the methods for loading spatial data**
- **Import dump files containing spatial data**
- **Create the control files associated with the bulk loading of spatial data using SQL*Loader**
- **Use the shapefile converter tool to convert an ESRI shapefile into a SQL*Loader control file**
- **Verify the validity of individual geometries, or of an entire layer**
- **Use the migration tool to convert from the relational model, or to migrate spatial data from 8.1.5 to the most recent version**

ORACLE

5-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

This lesson explains how spatial data is loaded into Oracle Spatial.

It describes how to load data by using standard Oracle utilities and commands (SQL*Loader, Import, Insert), as well as how to use a tool to move data from ESRI shapefiles into Oracle Spatial.

It also describes how to move data from the older relational model of Oracle Spatial into the object-relational model, and how to move data from the object-relational model used in earlier versions of Oracle Spatial into the latest Oracle Spatial format.

Loading Spatial Data

Two categories of loading:

- Bulk loading of data
 - SQL*Loader
 - Import
- Transactional inserts
 - INSERT statement

ORACLE

5-3

Copyright © Oracle Corporation, 2001. All rights reserved.

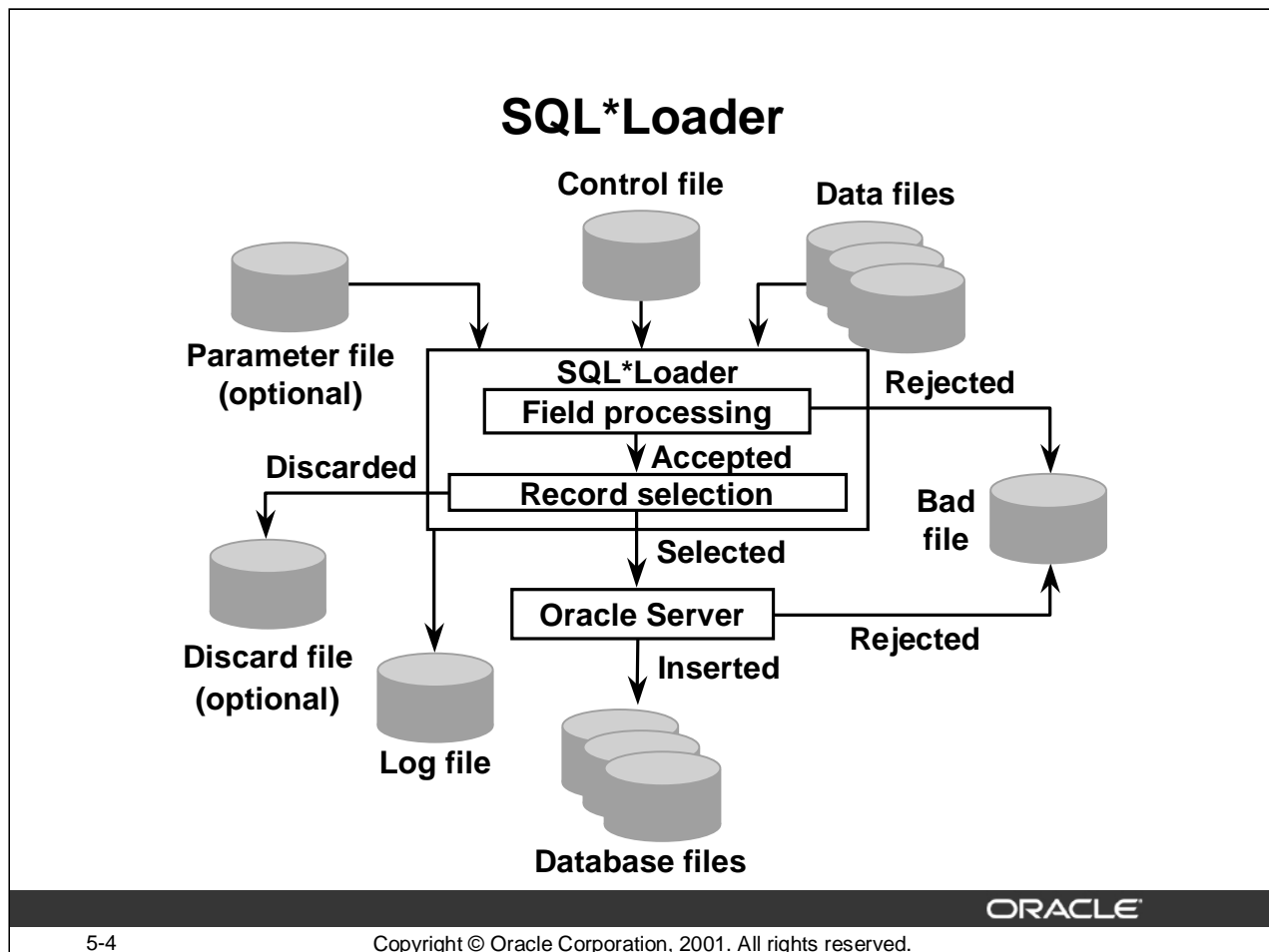
Using Geometric Data with Oracle Spatial

Two steps are involved in moving any data into a database so that it can be queried efficiently: loading the data tables, and creating or updating the index on the tables.

These same two steps apply to spatial data in the same way that they apply to other types of data.

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

- Bulk loading of data: This process is used to load large volumes of data into the database. If you are inserting data into the database, the SQL*Loader utility is used. If you are moving data between databases, then the data is exported in a bulk operation from one database, and imported into another database. There are no new concepts, features, or syntax associated with import/export of tables with spatial layers. The exact same import/export commands that move tables without spatial layers between databases also work to move tables with spatial layers between databases.
- 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.



Bulk Loading of Data with SQL*Loader

Bulk loading can be used to add large amounts of data (spatial and nonspatial) into Oracle. Bulk loading is accomplished by inserting data into the table by using the Oracle9i 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 upon 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 via a file.

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 also may produce a discard file containing records that did not meet the specified selection criteria.

See *Oracle9i Server Utilities* documentation for more information about SQL*Loader. The SQL*Loader command line executable name may differ based on operating system platform.

SQL*Loader Features

- One or more input files in a single load
- Input records can be combined into one logical record
- A single physical input record can be incorporated into multiple logical records
- Input fields can be fixed or variable length
- Input data can come from many formats: character, binary, date, packed and zoned decimal
- Data loaded from different types of media, for example, disk, tape, or named pipes
- Can load several tables in one run

ORACLE

5-5

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Loader Features

SQL*Loader can:

- Load data from multiple input data files of the same or different file types
- Combine multiple physical records into a single logical record so that a single row can be loaded from multiple physical rows in the load file
- Handle a single physical record as multiple logical records so that one input record in the load file can be used to put data into multiple rows in the table
- Handle fixed-format, delimited-format, and variable-length records. Fixed format is when you specify the exact format of the input file. Delimited format is the user chooses what the delimiter is between columns (a comma is a common delimiter). When not using fixed format, the size of the input record does not have to be specified, i.e. it is of variable length
- Support a wide range of data types, including date, binary, packed decimal, zoned decimal, numerics, and so on.
- Load data from disk, tape, or named pipes. When using named pipes under programmatic control, the input data is never actually written to disk as an input file
- Load multiple tables during the same run, loading selected rows into each table

SQL*Loader Features

- Can replace or append data to existing tables
- Can apply SQL functions to input data before it is stored
- Key values can be autogenerated
- Data can be loaded using high-performance “direct path” loads
- Uses the operating system
- Does error reporting

ORACLE

5-6

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Loader Features (continued)

Additional features of SQL*Loader include the ability to:

- Append data to the existing table, or replace data in the existing table
- Manipulate data fields with SQL functions before inserting the data into database columns
- Generate unique, sequential key values in specified columns
- Use high-performance “direct” loads to load data directly into database files with less internal Oracle processing
- Use the operating system's file or record management system to access data files
- Thoroughly report errors so that you can easily adjust and load all records

SQL*Loader Limitation

SQL*Loader cannot load directly from common spatial formats or vendor-specific data files

- **You can use third-party tools for this**
- **Tools are typically provided by the GIS/Spatial software vendor**
- **You can use the (unsupported) shapefile converter for ESRI shapefile format**

ORACLE

5-7

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Loader and Other Vendors' Spatial Data

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 FME from Safe Software (www.safe.com). There are others also.
- If the GIS/spatial software vendor supports their own format as well as Oracle Spatial format, there are usually tools provided by that vendor to move data from their format to Oracle Spatial format.
- Supplied as part of this course, and downloadable from OTN, is the Shapefile to Spatial converter (shp2sdo.exe). If your data can be extracted in ESRI shapefile format, the converter can create a SQL*Loader load file. SQL*Loader can then be invoked to put that spatial data into Oracle.

SQL*Loader for Point Data

```
LOAD DATA
  INFILE *
  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_POINT COLUMN OBJECT
      (X          FLOAT EXTERNAL,
       Y          FLOAT EXTERNAL)
    )
  )
  BEGINDATA
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

5-8

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Loader Control File for Loading Point Data

Above is an example of a SQL*Loader control file used for loading point data into Oracle Spatial. Note the following:

INFILE * means the data for the load will be in the control file, after the keyword BEGINDATA.

INTO TABLE cities means the cities table will 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 means to set to NULL any columns specified if the input record ends before all the fields have been filled.

Next comes each of the columns. NULLIF column=BLANKS means to set the column to NULL if the only thing being loaded in the column is blanks (spaces).

Next comes the location column of type SDO_GEOMETRY. The syntax shown (that is, the COLUMN OBJECT keywords) is specific for objects; the syntax isn't specific to Oracle Spatial.

The EXTERNAL keyword with the data type is required.

SQL*Loader Control File for Loading Point Data (continued)

Next comes the data after the BEGINDATA keyword. The CITY name field is loaded first, then the STATE_ABRV, then the city's population in 1990 (POP90), then the rank in terms of population (RANK90), and finally the SDO_GTYPE and SDO_POINT fields are loaded.

Note: Any parts of the SDO_GEOMETRY object that are not specified in the definition of what is to be loaded are set to NULL. In the example above, the SDO_SRID, SDO_ELEM_INFO, and SDO_ORDINATES fields are all set to NULL.

SQL*Loader for Lines and Polygons (Oracle9i)

```
LOAD DATA
INFILE *
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_ELEM_INFO VARRAY TERMINATED BY '/' (X FLOAT EXTERNAL),
     SDO_ORDINATES VARRAY TERMINATED BY '/' (X FLOAT EXTERNAL) ))
BEGINDATA
  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

5-10

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Loader Control File for Lines/Polygons (Oracle9i)

The above example is very similar to the previous example. There are some differences though. The first difference is:

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

This example is loading table counties, 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 the SDO_GEOMETRY are being loaded (the last example loaded the SDO_POINT field and left the SDO_ELEM_INFO and SDO_ORDINATES as NULL). There is special syntax to end each of the varray types (each value within the varray is terminated with a vertical bar, but the varray itself needs to be terminated also). The syntax:

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

is the way to describe to SQL*Loader that the SDO_ELEM_INFO varray ends with '/'. The same syntax is shown for the SDO_ORDINATES varray.

SQL*Loader for Lines and Polygons (Oracle 8i)

```
LOAD DATA
  INFILE *
  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_ELEM_INFO  VARRAY TERMINATED BY '|'/' (X FLOAT EXTERNAL),
       SDO_ORDINATES  VARRAY TERMINATED BY '|'/' (X FLOAT EXTERNAL) ))
  BEGINDATA
  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|....|/
```

Note: #+ is necessary when the ordinate array spans more than 1 line (bug #1078976)

ORACLE

5-11

Copyright © Oracle Corporation, 2001. All rights reserved.

SQL*Loader Control File for Lines/Polygons (Oracle8i)

The above example is the same as the previous example with one important difference.

Note after the data begins after the BEGINDATA keyword there is a single line with #+.

Due to a bug in Oracle8i this line is required when a VARRAY spans more than one physical record. As you will see later, one of the reasons a logical record might span more than one physical record is due to a limitation in SQL*Loader. SQL*Loader cannot process physical records larger than 64 KB characters. This workaround can only be applied to Oracle8i SQL*Loader control files. The #+ syntax is not acceptable in Oracle9i.

SQL*Loader Restrictions

- **SQL*Loader does not support *direct path* mode when loading the VARRAY type.**
 - Use conventional path if loading SDO_ELEM_INFO and SDO_ORDINATES
 - Direct path is supported if only loading SDO_POINT field
- **SQL*Loader cannot process physical records larger than 64 KB.**
 - Break the input into multiple physical records (see the example shown previously)
- **SQL*Loader and spatial indexes:**
 - Spatial index is maintained during conventional path load
 - Spatial index is not maintained during direct path load

ORACLE

5-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Restrictions

Keep in mind a few restrictions when loading a spatial layer using SQL*Loader:

In Oracle9i Release 1, SQL*Loader does not support loading of VARRAYs using direct path mode. Only conventional path load is supported if the VARRAY columns (SDO_ELEM_INFO and SDO_ORDINATES) are loaded. If the SDO_POINT field is loaded and the SDO_ELEM_INFO and SDO_ORDINATES fields are not loaded, then direct path mode can be used to load spatial data.

SQL*Loader will not process physical records longer than 64K characters long. If a physical record will be greater than 64 KB it needs to be broken into two or more physical records. The previous example shows how to break physical records up using the CONTINUEIF syntax.

Spatial indexes are maintained when using conventional path mode in SQL*Loader. This can slow the bulk loading of spatial data significantly. When bulk loading large amounts of data, Oracle recommends dropping the spatial index before the load, and creating the spatial index after the load. When using direct path mode of SQL*Loader, spatial indexes are not maintained. Oracle recommends dropping the spatial index before using SQL*Loader in direct path mode, and creating the spatial index when the load is completed.

Export/Import

- **Export**
 - Used for bulk unloading of Oracle tables
 - No special syntax for spatial data
 - Indexed spatial data
 - Moves associated geometry metadata
- **Import**
 - Used for bulk loading previously exported tables
 - No special syntax
 - Indexed spatial data
 - Adds user_sdo_geom_metadata entry
 - Rebuilds index using same command as originally specified

ORACLE

5-13

Copyright © Oracle Corporation, 2001. All rights reserved.

Export/Import

Export and import are used for bulk unloading (export) and loading (import) of Oracle tables. There is no Oracle Spatial-specific syntax required for exporting tables with spatial layers.

Export creates a file (called a dump file) with the contents of whatever tables were specified on the export command. By default, export will unload all of the data associated with a user's schema. By specifying the tables parameter, the contents of one or more specific tables can be unloaded. If a spatial table is not spatially indexed, export will unload the data for the table only. If the table is spatially indexed, export will also unload the contents of the user_sdo_geom_metadata view associated with that table, as well as the command used to build the index.

Import is used to add data that was previously exported into an Oracle database. If the export file contains spatial data that was indexed, an entry is added to the user_sdo_geom_metadata view and the index is rebuilt using the same command the index was originally built with. If the spatial data was not indexed, the metadata information will have to be added outside of the context of the import.

For more information about export and import, see the *Oracle9i Database Utilities* documentation.

Export/Import (continued)

Note: Oracle allows data to be exported by one user and imported into another user's schema. If the data was exported from a privileged user's schema, it cannot be imported into a nonprivileged user's schema without using a privileged account to facilitate the transfer. When moving indexed spatial data in this manner in pre-9i versions of Oracle, the `user_sdo_geom_metadata` associated with the data was associated with the privileged account that facilitated the transfer rather than the target user's schema. This has been fixed in a patch for Oracle 8.1.7, and in Oracle9i.

Transactional Inserts

- Accomplished using a standard SQL insert statement
- Used to add geometric features to an existing spatial layer
- Spatial index automatically updated

```
SQL> INSERT INTO LINES VALUES (  
2>     attribute_1, ... attribute_n,  
3>     MDSYS.SDO_GEOMETRY (  
4>         2002, null, null,  
5>         MDSYS.SDO_ELEM_INFO_ARRAY (1,2,1),  
6>         MDSYS.SDO_ORDINATE_ARRAY (  
7>             10,10, 20,25, 30,10, 40,10))  
8> );
```

ORACLE

5-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Transactional Inserts

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

The INSERT statement is used to add spatial data. See the Oracle9i SQL Reference for information on INSERT.

Note the use of the geometry constructor, which was introduced in the “Creating Spatial Layers” lesson. You should start becoming more familiar with its use. In this case, the geometry constructor is used to create a geometry object, and inside that constructor 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, that index is maintained automatically when new data is inserted into the table (as it is for updates and deletes). There is much more information about Spatial Indexes coming up in later chapters devoted to this topic.

Transactional Inserts

INSERT limited to 999 values in SDO_ORDINATES field when using the SDO_GEOMETRY constructor directly. Use host variable for the geometry object instead.

```
CREATE OR REPLACE PROCEDURE
    INSERT_GEOM(GEOM MDSYS.SDO_GEOMETRY)
IS
BEGIN
    INSERT INTO TEST_1 VALUES (GEOM);
    COMMIT;
END;
/
```

ORACLE

5-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Insert Limitation with Constructors

When using 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 arrays then you can load data from within a program or use PL/SQL to load data.

The above example shows how to create a PL/SQL procedure that will insert into a table using a variable. On the next page we'll see how to populate data into the variable, and call the procedure.

Transactional Inserts

```
declare
geom MDSYS.SDO_GEOMETRY :=
MDSYS.SDO_GEOMETRY (2003, null, null,
                    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3),
                    MDSYS.SDO_ORDINATE_ARRAY (-109,37,-102,40));
BEGIN
  INSERT_GEOM(GEOM);
END;
/
```

Variable can contain over 999 values

ORACLE

5-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Insert Limitation with Constructors (continued)

The previous example defined the INSERT_GEOM procedure. In this example the use of an anonymous PL/SQL block which calls the INSERT_GEOM procedure is shown.

First, a variable of type SDO_GEOMETRY is defined. In the same statement as its type definition the value is assigned (the value could have been assigned in the body of the procedure instead). Note that the SDO_ORDINATES or SDO_ELEM_INFO arrays could have been initialized with greater than 999 values.

Shapefile Converter

- ***Unsupported tool***
- **Processes one ESRI shapefile at a time**
- **Processes the attributes (.DBF) as well as the geometries (.SHP and .SHX)**
- **Generates a SQL script for creating the spatial table and loading spatial metadata**
- **Generates a SQL*Loader control file**
- **Uses continuation marks to break large records into pieces acceptable by SQL*Loader**

ORACLE

5-18

Copyright © Oracle Corporation, 2001. All rights reserved.

Shapefile Converter

The shapefile converter is an unsupported tool available both as part of this course and as a download from OTN that converts ESRI shapefiles (geometries and attributes) into SQL*Loader control and data files.

The shapefile converter automatically generates a SQL script that can be used to create the single table that holds both geometry and attribute data. Additionally, the script also inserts the metadata about the spatial layer into USER_SDO_GEOM_METADATA.

The shapefile converter appropriately uses continuation marks to ensure no physical record is greater than 64K characters. It can convert data for loading into Oracle Spatial 8.1.6 and beyond.

After the shapefile converter is used, the generated SQL script must be executed to create the table and add metadata to the USER_SDO_GEOM_METADATA view. The script can be edited to add storage parameters to the create table statement. After that, SQL*Loader must be invoked to bulk load data into the database.

When the load is complete, the data is migrated to the latest Oracle Spatial format using the SDO_MIGRATE.TO_CURRENT procedure supplied as part of Oracle Spatial. This procedure makes sure all elements have 4 digit SDO_GTYPEs, four digit SDO_ETYPEs where appropriate, takes care of ring ordering and ensures each exterior polygon is followed by all of its interior polygons. The next slides describe in detail how to use the shapefile converter.

Invoking the Converter

```
shp2sdo <shapefile> <table-name> -g <geometry-col>  
-i <id-col> -n -d -x (<Xmin>,<Xmax>)  
-y (<Ymin>,<Ymax>) -s <srid> -t <tolerance>
```

- **<shapefile>** = name of the shapefile to convert (do not include the .SHP suffix)
- **<table-name>** = name of the output table and prefix for the generated output files
- **-g <geometry-col>** = name of the geometry column (layer name) in the output table
- **-i <id-col>** = name of the id column
- **-n [<start_id>]** = number to start id-col (default 0)
- **-d** = include the data in the control file

ORACLE

5-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking the SHP2SDO Converter

This example shows how to invoke the shapefile to Oracle Spatial converter.

The first parameter is the name of the shapefile. The converter will use input from the .shp file as well as the .dbf and .shx files.

The second parameter is the name of the spatial table to create, as well as the prefix for the output files generated as a result of running the shapefile converter. If not specified, the table name is the same as the shapefile.

Next in this example is the -g <geometry-col> parameters, which specify the name of the geometry column to create in the table specified as the second parameter. This column will be of type MDSYS.SDO_GEOMETRY.

-i <id-col> is an optional parameter that creates a numeric column that can be used to store an id field. Oracle Spatial has no requirement that this field exist, but many tools do require a numeric key field.

-n [<start-id>] is a parameter used to populate the numeric key field created by the -i qualifier. If a start-id is provided the numbers are populated starting at this value. Otherwise, the numbers start at 0.

The -d parameter is used to specify that the data should be included directly into the control file. Otherwise, a separate data file is used.

Invoking the Converter

- **-x (Xmin, Xmax) = layer dimensions for the first dimension. If not specified, then the actual bounds of the shapefile are used**
- **-y (Ymin, Ymax) = layer dimensions for the second dimension. If not specified, then the actual bounds of the shapefile are used**
- **-s <srid> = the srid to include for each geometry and in the layer metadata. If not specified, NULL**
- **-t <tolerance> = the tolerance to specify for the layer. If not specified, the default is 0.00000005. This value should be specified (do not use the default) if loading geodetic data**

ORACLE

5-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Invoking the SHP2SDO Converter (continued)

The -x (Xmin, Xmax) is used to tell Oracle the bounds (or extent) of the first dimension of the coordinate system used. If not specified, the bounds are extracted from the actual shapefile. Oracle recommends using this parameter to specify the largest bounds of the coordinate system. For data stored as Longitude/Latitude, the values of Xmin and Xmax should be set to the smallest and largest Longitude values, that is, -180, 180.

The -y (Ymin, Ymax) is used to tell Oracle the bounds (or extent) of the second dimension of the coordinate system used. If not specified, the bounds are extracted from the actual shapefile. Oracle recommends using this parameter to specify the largest bounds of the coordinate system. For data stored as Longitude/Latitude, the values should be set to the smallest and largest Latitude values, that is, -90, 90.

The -s <srid> is used to specify the coordinate system (spatial reference system) associated with the data. By default, no coordinate system information is loaded (NULL for SDO_SRID in the SDO_GEOMETRY column and NULL for SRID in USER_SDO_GEOM_METADATA).

Invoking the SHP2SDO Converter (continued)

-t <tolerance> should be specified if the tolerance value to be loaded in the DIMINFO array is a value other than 0.00000005. Tolerance the distance two numbers must be from each other not to be considered the same number. For Geodetic data (data represented in longitude/latitude), this value is specified in meters (that is, 0.5 is a one-half meter tolerance). For projected data (not longitude/latitude data, or no SRID set) this value is in the same units as the data. Match the decimal precision of ordinates with zeros and append a 5 (that is, 0.00000005).

Note

- In Oracle9i and beyond, for geodetic data, try not to use a smaller tolerance value than is necessary, because this causes additional precision requirements for all spatial operations, which may result in slower performance.
- Pre Oracle9i, for geodetic data, match the decimal precision of ordinates with zeros and append a 5 (that is, 0.00000005).

Example Run

- **shp2sdo cities -g location -x(-180,180) -y(-90,90)**
- **Input**
 - Files **CITIES.SHP, CITIES.SHX, CITIES.DBF**
- **Outputs**
 - **CITIES.SQL: creates the 'CITIES' table and loads CITIES spatial metadata**
 - **CITIES.CTL: SQL*Loader control file for loading the CITIES table.**
 - **CITIES.DAT: data file**

ORACLE

5-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Example SHP2SDO Converter Run

On the command line the first parameter (“cities”) is the name of the shapefile, so cities.shp, cities.shx, and cities.dbf will be processed from the current directory. Full path names to the shapefile can be specified. Because <table-name> is not specified as the second parameter, the name of the table to be created will also be “cities”. The name of the column in table cities of type MDSYS.SDO_GEOMETRY will be “location”, as specified by the -g qualifier.

The output files of the command are CITIES.SQL, CITIES.CTL, and CITIES.DAT. CITIES.SQL includes the command to create the cities table and the insert statement to load the USER_SDO_GEOM_METADATA view. The -x (-180,180) -y (-90,90) populates the DIMINFO field of USER_SDO_GEOM_METADATA with coordinate system bounds of -180,180 and -90,90. CITIES.CTL includes the information for SQL*Loader to load the cities table. CITIES.DAT contains the data to be loaded in the cities table.

After running the converter, execute cities.sql from SQL*Plus to create the table and load the USER_SDO_GEOM_METADATA view with information about the spatial layer. Next run SQL*Loader to load the table. Then migrate the data loaded to the latest Oracle Spatial format, and create a spatial index on the SDO_GEOMETRY column. The migration of the data will be covered later in this chapter, and information about spatial indexing will follow in a later chapter.

Log Output

```
shp2sdo - Shapefile(r) To Oracle Spatial Converter
Version 2.9 23-May-2001
Copyright 1997,1998,1999,2000,2001 Oracle Corporation

Processing shapefile cities into spatial table CITIES
Data model is object-relational
  Geometry column is LOCATION
  Points stored in SDO_POINT attributes
  Data is in a separate file(s)
  Bounds set to X=[-180.000000,180.000000]
                Y=[-90.000000,90.000000]
Conversion complete : 195 points processed
The following files have been created:
  cities.sql : SQL script to create the table
  citiesctl : Control file for loading the table
  cities.dat : Data file
```

ORACLE

5-23

Copyright © Oracle Corporation, 2001. All rights reserved.

Log Output from SHP2SDO

Above is the log output from running the shapefile to Oracle Spatial conversion utility. The output shows the user that the input is coming from the cities shapefile, the output table will be called CITIES, and data is created for the Oracle Spatial object-relational format. Additionally, the SDO_GEOMETRY column in table CITIES is called LOCATION, and point data will be stored in the SDO_POINT field. The data will be stored in a separate file. The extent of the coordinate system will be set to -180,180 and -90,90 in the the DIMINFO field of the USER_SDO_GEOM_METADATA entry associated with the LOCATION column in the CITIES table.

The output also shows that 195 cities are converted, and the output files are cities.sql, citiesctl, and cities.dat.

Usage Notes

- **Use -s to set the Spatial Reference System ID (SRID)**
- **By default, extents in the USER_SDO_GEOM_METADATA view are set to the bounds computed from the input shapefile: use the -x and -y options to override**
 - **Note: especially important for geodetic data**
- **Point data is loaded in the SDO_POINT field**
- **Use -d to force the data inside the control file**
- **Use -h for help**
- **Can also let the program prompt for parameters**
- **Use -t to set the tolerance**
 - **Note: This is especially important for geodetic data.**

ORACLE

5-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Usage Notes

As will be discussed in future lessons, to take advantage of the full functionality of all the new features in Oracle Spatial, it is important to know about the data that is loaded. If the data is geodetic, it is important to set a Spatial Reference System ID (SRID) associated with geodetic data. If the data is projected, it is important to include the SRID of the projection. This gives maximum flexibility and functionality when using the data with the Oracle Spatial product.

For geodetic data, the bounds of the coordinate system should always be set to -180,180 and -90, 90. For quadtree indexes, it is critical that all geometry ordinates fall in the range of SDO_LB to SDO_UB for each dimension, and recommended to have the same upper and lower bounds for all spatial data in a project. R-tree indexes are more flexible, and can accommodate geometries that fall outside of the SDO_LB to SDO_UB range.

Point data is always appropriately loaded in the SDO_POINT field.

The -d qualifier is useful to keep the SQL*Loader data and the control information in a single file.

Usage Notes (continued)

Use `shp2sdo -h` for help.

If `shp2sdo` with no parameters is typed on the command line, the program prompts for input. `-t <tolerance>` should be specified if the tolerance value to be loaded in the DIMINFO array is a value other than 0.00000005. Tolerance the distance two numbers must be from each other not to be considered the same number. For Geodetic data (data represented in longitude/latitude), this value is specified in meters (that is, 0.5 is a one half meter tolerance). For projected data (not longitude/latitude data, or no SRID set) this value is in the same units as the data. Match the decimal precision of ordinates with zeros and append a 5 (that is, 0.00000005).

Note

- In Oracle9i and beyond, for geodetic data, try not to use a smaller tolerance value than is necessary, because this causes additional precision requirements for all spatial operations, which may result in slower performance.
- Pre Oracle9i, for geodetic data, match the decimal precision of ordinates with zeros and append a 5 (that is, 0.00000005).

Interactive Run

```
shp2sdo - Shapefile(r) To Oracle Spatial Converter
Version 2.9 23-May-2001
Copyright 1997,1998,1999,2000,2001 Oracle Corporation

Input shapefile (no extension): states
  Shape file states.shp contains 56 polygons
Output table [states]: states
Output data model [O]:
Geometry column [GEOM]:
ID column []:
Use a spatial reference system ID (SRID) ? [N]: Y
Please enter an SRID value: 8307
Change tolerance value from the default (0.00000005) ? [N]: Y
Please enter a new tolerance value: 0.5
Generate data inside control files ? [N]: Y
Bounds: X=[-179.144806,179.764160] Y=[-14.605210,71.332649]
Override ? [N]: Y
Xmin [-179.144806]: -180
Xmax [179.764160]: 180
Ymin [-14.605210]: -90
Ymax [71.332649]: 90
Processing shapefile states into spatial table STATES
Data model is object-relational
  Geometry column is GEOM
  Points stored in SDO_POINT attributes
  Data is in the control file(s)
  Bounds set to X=[-180.000000,180.000000] Y=[-90.000000,90.000000]
Conversion complete : 56 polygons processed
The following files have been created:
  states.sql : SQL script to create the table
  states.ctl : Control file for loading the table
```

ORACLE

5-26

Copyright © Oracle Corporation, 2001. All rights reserved.

An Interactive Run of the Shapefile to Oracle Spatial Converter

Above is an example of an interactive run of the Shapefile to Oracle Spatial converter. Information input by the user is shown on the right side of the colon (:).

The states shapefile is the input, the output table is states, the data model is object-relational, and the default column name of GEOM will be used to hold the column of type MDSYS.SDO_GEOMETRY. No numeric key field (ID column) will be loaded. The data will be loaded with an SRID of 8307, which specifies the geodetic coordinate system called WGS 84.

Additional information specified includes a nondefault tolerance value of 0.5 (meters is the default units for geodetic data), and nondefault coordinate system bounds of -180 to 180 in the first dimension (Longitude) and -90 to 90 in the second dimension (Latitude).

Validating Geometries

- **SDO_GEOM.VALIDATE_GEOMETRY**
 - Determines if a geometry is valid
- **SDO_GEOM.VALIDATE_LAYER**
 - Determines if all the geometries stored in a column are valid

ORACLE

5-27

Copyright © Oracle Corporation, 2001. All rights reserved.

Validating Geometries

Oracle Spatial provides two functions to validate geometries.

- The first function, **SDO_GEOM.VALIDATE_GEOMETRY**, determines if a geometry is valid.
- The second function, **SDO_GEOM.VALIDATE_LAYER**, determines if all the geometries stored in a layer are valid.

These functions perform type consistency, for example, whether the **SDO_GTYPE** valid, whether the **SDO_ETYPE** is consistent with the **SDO_GTYPE**, and whether the **SDO_ELEM_INFO** varray contains triplets.

The functions also perform geometry consistency operations, such as making sure the first and last points of a polygon are the same, that a polygon does not cross itself, that points on a line are distinct and nonrepeating, that line strings have at least two points, along with many other checks.

For more information regarding all of the checks done by the validation functions, see the *Oracle Spatial User's Guide and Reference*.

The VALIDATE_GEOMETRY Function

```
error := SDO_GEOM.VALIDATE_GEOMETRY  
( <geometry>,  
  <diminfo> )
```

- **<geometry> = SDO_GEOMETRY that holds the object to verify**
 - can be a variable or table column
- **<diminfo> = dimensions array**
- **returns = TRUE if valid, an Oracle error number if invalid for a known reason, or FALSE if invalid for unknown reason**

ORACLE

5-28

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_GEOM.VALIDATE_GEOMETRY Function

Input to the SDO_GEOM.VALIDATE_GEOMETRY function includes the geometry to validate and the diminfo array associated with the layer the geometry is in.

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

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

If the geometry is invalid, the returned value is a standard Oracle error number that specifies the reason the geometry is invalid, or FALSE if the reason cannot be specified.

VALIDATE_GEOMETRY Example

```
SQL> SELECT SDO_GEOM.VALIDATE_GEOMETRY
2   (s.geom,
3     (SELECT diminfo
4       FROM user_sdo_geom_metadata
5       WHERE table_name = 'PROJ_STATES'
6         AND column_name = 'GEOM')) STATUS
7   FROM proj_states s
8   WHERE s.state = 'New Jersey';
STATUS
-----
TRUE
```

ORACLE

5-29

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.VALIDATE_GEOMETRY Example

This query is an example of running SDO_GEOM.VALIDATE_GEOMETRY.

First, the query selects the GEOM column from the STATES table where the geometry is associated with the state of 'New Jersey' and passes it into the SDO_GEOM.VALIDATE_GEOMETRY function.

Additionally, the diminfo array associated with the GEOM column of the STATES table is obtained from the USER_SDO_GEOM_METADATA view and included in the function.

Following the function call is the word STATUS, which is used as a column alias when printing out the results of the function call.

In this case, the function returns the value TRUE, which means that the geometry associated with New Jersey is valid.

The VALIDATE_LAYER Procedure

```
SDO_GEOM.VALIDATE_LAYER  
( <table-name>, <column-name>, <key-column>,  
  <result-table> [, <commit-interval>])
```

- **<table-name> and <column-name> identify the layer to verify**
- **<key-column> is the name of the primary key of that table**
 - **Must be a *number*!**
- **<result-table> is the name of a table that will receive the results of the validation**
- **<commit-interval> is the frequency that results will commit or be written to the <result-table>**

ORACLE

5-30

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_GEOM.VALIDATE_LAYER Procedure

The SDO_GEOM.VALIDATE_LAYER procedure validates an entire layer at a time. This procedure requires that the table with the layer to be validated contain a numeric primary key value. VALIDATE_LAYER also requires the <result-table> be created before it is called.

Input to the SDO_GEOM.VALIDATE_LAYER procedure includes:

<table-name>: The table name that contains that spatial layer to be validated.

<column-name>: The column name of type SDO_GEOMETRY that contains the spatial layer.

<key-column>: The numeric primary key associated with the table.

<result-table>: The name of the table that the results of the function are written to. The table must be created prior to running the function. The next slide has more information about the <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. Every <commit-interval> geometries, a progress status is inserted into the result table (that is, the <key-column> value and "DONE"). If no <commit-interval> is specified, the default is to commit after the entire layer has been validated.

SDO_GEOM.VALIDATE_LAYER only reports invalid geometries to the result table, or a progress status.

The Results Table

- **Must be manually created prior to using the procedure**

```
CREATE TABLE VALIDATION_RESULTS (  
  OBJECT_ID    NUMBER,  
  STATUS       VARCHAR2(10)  
);
```

- **You define table name**
- **You define column names in data type order**

ORACLE

5-31

Copyright © Oracle Corporation, 2001. All rights reserved.

The Results Table

The table that holds the results of the SDO_GEOM.VALIDATE_LAYER procedure must exist before the procedure is run.

The name of the table as well as the names of each of the columns is user-defined. Oracle Spatial requires that the columns in the table be of type number and of type VARCHAR2(10). The columns must be created in that order, that is, the first column must be of type number, and the second must be of type VARCHAR2(10).

VALIDATE_LAYER Example

```
SQL> begin
  2   SDO_GEOM.VALIDATE_LAYER
  3   ('RIVERS','GEOM','GID','VALIDATION_RESULTS');
  3   end;
  4   /
SQL> SELECT * FROM validation_results;
OBJECT_ID STATUS
-----
2 13341
1005 13356
1009 13356
1010 13356
1011 13356
```

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

ORACLE

SDO_GEOM.VALIDATE_LAYER EXAMPLE

This example executes the SDO_GEOM.VALIDATE_LAYER procedure. As can be seen, all of the geometry objects in column GEOM in the RIVERS table are validated. The numeric primary key in table RIVERS is GID. The results of the procedure are stored in table VALIDATION_RESULTS. Since 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 call to SDO_GEOM.VALIDATE_LAYER shows the results stored in the VALIDATION_RESULTS table. The data in the OBJECT_ID column is the GID associated with the geometry in the original table, and the data in the STATUS column is the error code associated with that geometry.

In this example, the geometry with GID=2 in the RIVERS table has an error code of 13341, which means the line geometry has less than two coordinates. At the other locations, the error code is 13356 which means that adjacent points in the line string are redundant.

Error codes can be found in the *Oracle9i Database Error Messages* document.

VALIDATE_GEOMETRY Example Similar to VALIDATE_LAYER

Useful to validate a layer that has no numeric key

```
CREATE TABLE validation_results as
  SELECT STATE,
         SDO_GEOM.VALIDATE_GEOMETRY(s.geom,
         (SELECT diminfo
          FROM user_sdo_geom_metadata
          WHERE table_name = 'PROJ_STATES'
           AND column_name = 'GEOM')) STATUS
  FROM proj_states s;

SELECT * FROM validation_results
WHERE status <> 'TRUE';
```

ORACLE

5-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Using VALIDATE_GEOMETRY to Validate a Layer

This is an example of using SDO_GEOM.VALIDATE_GEOMETRY to validate an entire layer. This is useful in cases when there is no numeric key column in the layer, but a layer validation is desired.

This statement creates a table called validation_results that contains two columns. The first column, STATUS, contains the result of the validation, and the second column, STATE, contains the STATE name from the STATES table.

The statement validates all of the geometries in the STATES layer.

Every result is stored, so if the geometry was valid then a status of TRUE is stored. If the geometry is invalid, then the error code is stored.

The select statement on the bottom selects all of the data from the validation_results table where the status indicates an invalid geometry.

This example could be modified to be a PL/SQL script that only writes invalid geometries to the validation_results table, and to commit periodically.

Spatial Migration Utility (SDO_MIGRATE.TO_CURRENT)

Same utility can:

- **Migrate data from the Oracle Spatial relational schema to the Oracle9i Object-Relational model**
 - Before Oracle9i this functionality in procedure **SDO_MIGRATE.TO_81X**
- **Migrate data from the 8.1.5 object-relational format to the Oracle9i Object-Relational model**
 - Before Oracle9i this functionality in procedure **SDO_MIGRATE.FROM_815_TO_81X**

ORACLE

5-34

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_MIGRATE.TO_CURRENT

Prior to objects in Oracle (that is, in Oracle 7), Oracle Spatial stored geometry coordinates in a relational table format. In Oracle 8i (that is, in Oracle 8.1.5), the SDO_GEOMETRY object was introduced to store spatial data. In Oracle 8.1.6, a four digit SDO_GTYPE and four digit element type for polygons was introduced. In Oracle 8.1.6, polygon rotation rules were also introduced.

The SDO_MIGRATE.TO_CURRENT can:

- Migrate data from the relational schema to the latest object-relational format
- Migrate data from the object-relational schema introduced in 8.1.5 to the latest object-relational format.

Spatial Migration Utility (Relational Schema to Current)

- **Relational model had four tables for each layer**
 - **LAYER_SDOLAYER**
 - **LAYER_SDODIM**
 - **LAYER_SDOGEOM**
 - **LAYER_SDOINDEX**
- **Only tables you migrate are LAYER_SDOGEOM and LAYER_SDODIM**
- **Geometries could span rows**
 - **Attributes stored in a separate table**
 - **Join using SDO_GID**

ORACLE

5-35

Copyright © Oracle Corporation, 2001. All rights reserved.

Relational to Object-Relational Migration

Only spend time on this slide and the two slides following this slide if you think you may need to migrate data from the old spatial format using the relational model to the new spatial format using the object-relational model. The relational model of Oracle Spatial is in the process of being deprecated, hence all spatial data should be migrated to the object-relational model.

In the relational model, there were four different tables used to build a spatial layer.

- **LAYER_SDOLAYER:** This table holds information about the geometries stored in the layer, such as the number of ordinates that can be stored in each row that makes up the geometry
- **LAYER_SDODIM:** For each dimension, values such as upper and lower bounds as well as tolerance are stored.
- **LAYER_SDOGEOM:** The values of the ordinates that make up the geometry are stored here
- **LAYER_SDOINDEX:** This table contains index information.

At least one additional table is used to store the attribute data associated with the spatial layer. The only tables migrated during the SDO_MIGRATE.TO_CURRENT procedure are LAYER_SDOGEOM and LAYER_SDODIM.

Relational to Object-Relational Migration (continued)

In the Oracle Spatial relational model, the LAYER_SDOGEOM table was defined on a layer-by-layer basis with enough columns to hold a given number of coordinates. If the geometry was larger than the number of coordinates allowed in each row, the geometry coordinates were also loaded into the next row. In this way, a single geometry might require many rows to store all of its coordinates. Oracle Spatial uses keys to associate values in the LAYER_SDOGEOM table with a single geometry. After migrating the data, the attribute values can be populated into the new table by joining on the key field SDO_GID, which was required in the relational model.

Data is also loaded into the USER_SDO_GEOM_METADATA view during the migration.

To complete the migration, a spatial index can be created (a later chapter will discuss spatial indexes).

Spatial Migration Utility (Relational Schema to Current)

- Migrates a relational schema to an object-relational schema
- Migrates data and metadata

```
SDO_MIGRATE.TO_CURRENT(  
  LayerName, TableName, GidColName, GeomColName,  
  [LayerGTYPE], [UpdateFlag])
```

- LayerName: Name of layer in relational schema
- TableName: Name of table that will contain the SDO_GEOMETRY object
- GidColName: SDO_GID column in relational schema will be renamed to GidColName in the new table
- GeomColName: Name of the column that will contain the SDO_GEOMETRY object
- LayerGTYPE: If you are migrating point only data, set to 'POINT' for optimal migration performance. Default is NOTPOINT
- UpdateFlag: If you are migrating into an existing populated attribute table, set this parameter to 'UPDATE', default is 'INSERT'

ORACLE

5-37

Copyright © Oracle Corporation, 2001. All rights reserved.

Migrating from the Relational Model to the Object-Relational Model

The procedure that migrates Oracle Spatial tables from the relational format to the object-relational format is found in the SDO_MIGRATE package included with Oracle Spatial, and is called TO_CURRENT. The TO_CURRENT procedure works in two steps. In the first step, spatial data is migrated from the relational model to the object-relational model. In the next step, the data is updated to ensure conformance with the latest object-relational model. The data is automatically changed so that rules about polygon rotation and order are adhered to, as well as changes associated with the use of four digit SDO_GTYPEs and four digit element types (used for inner and outer polygons).

Migrating from the Relational Model to the Object-Relational Model (continued)

The same procedure also has a different set of parameters that will not do the relational to object-relational migration, but will migrate object-relational spatial data from the 8.1.5 format to the 8.1.6 format. Changes are made to the data so that it adheres to the rules about polygon rotation and ring ordering. Also the data is updated to the four digit SDO_GTYPEs and four digit element types.

The parameters for the call include the LayerName, which is the relational model layer name, the TableName, which is the name of the table that will be inserted into or updated to contain the spatial layer, the GidColName, which is the name of the numeric key column associated with each geometry in the relational model, and the GeomColName, which is the name of the column of type SDO_GEOMETRY that will contain the migrated data.

Additional optional parameters include LayerGTYPE, which can be set to point for optimal migration performance, and an UpdateFlag, which should be set to 'UPDATE' if the geometry data is to be populated into an existing attribute table.

Migration Example (Relational Schema To Current)

```
-- Add the geometry column
ALTER TABLE CITIES_ATT ADD LOCATION MDSYS.SDO_GEOMETRY;

-- Migrate the geometries (commits automatically)
EXECUTE MDSYS.SDO_MIGRATE.TO_CURRENT ('CITIES', 'CITIES_ATT',
'SDO_GID', 'LOCATION', 'POINT', 'UPDATE');

-- Drop the SDO_GID column
ALTER TABLE CITIES_ATT DROP COLUMN SDO_GID;
```

- Add MDSYS.SDO_GEOMETRY column to attribute table
- Migrate using the UPDATE flag
- If the SDO_GID column not required, it can be dropped

ORACLE

5-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Migration Example

In this example, the CITIES layer is migrated from the relational model to the object-relational model. This is accomplished by altering the existing attributes table associated with the CITIES layer (CITIES_ATT) by adding a column called LOCATION, which is of type MDSYS.SDO_GEOMETRY.

Next, the migration procedure SDO_MIGRATE.TO_CURRENT is invoked. The layer name is CITIES; the table name to migrate the layer into is called CITIES_ATT; and the key field to match the attributes to the geometry is called SDO_GID. The geometry column is called LOCATION; and the data type of the geometries is exclusively point data. Instead of creating a new table with a new spatial layer the procedure will update the existing table, matching the geometry data in the rows where the SDO_GID matches.

The last step is drop the SDO_GID column if it is not needed. Note that some tools require a numeric key value in the data, so check with your tool vendor before dropping this column.

Data Migration

8.1.5 SDO_GEOMETRY to Current

- Use SDO_MIGRATE.TO_CURRENT

```
SQL> execute SDO_MIGRATE.TO_CURRENT('counties','geom',1000);
```

- Will convert the SDO_GTYPEs
- Will convert the polygons to the new notation:
 - Element types (1003, 2003, 1005, 2005)
 - Element ordering (outer then inner rings)
 - Polygon rotation
- Includes commit interval for migration
- *Cannot rollback*

ORACLE

5-40

Copyright © Oracle Corporation, 2001. All rights reserved.

Migrating Object-Relational Data

The same procedure used to migrate spatial data from the relational to the object-relational model of Oracle Spatial can be used to migrate object-relational data from the 8.1.5 data format of Oracle Spatial to the latest format. Once the data has been migrated from the 8.1.5 model, it does not have to be migrated again because the model has not changed. The migration routine does the following:

- Converts SDO_GTYPEs from single digits to four digits, encoding the number of dimensions directly into the SDO_GTYPE
- Converts to the new four digit element types to encode polygons, using 1003 and 1005 for external polygons, and 2003 and 2005 for inner polygons
- Changes element ordering of polygons, ensuring all inner polygons directly follow the exterior polygon they are inside
- Changes coordinate ordering to ensure exterior rings are counter-clockwise, and interior rings are clockwise

In this example, a commit interval of 1000 is specified, causing the migration to commit after every 1000 geometries. Using the commit interval allows the migration process to complete on a large layer without requiring a large rollback segment.

Note the migration process is a one-way process, and it cannot be rolled back.

Manual Data Migration

8.1.5 to 8.1.6 and Higher

For points and lines, can manually convert SDO_GTYPEs:

```
SQL> update cities x
      2  set x.location.sdo_gtype =
      3      x.location.sdo_gtype + 2000;
195 rows updated.
SQL> commit;
```

- Both methods (manual and using migration tool) will trigger an update of the spatial index if it exists.
- Oracle recommends dropping spatial indexes before data migration, and re-creating the indexes after migration.

ORACLE

5-41

Copyright © Oracle Corporation, 2001. All rights reserved.

Manual Process to Migrate Data

The only changes required for data that does not include polygons is to update the SDO_GTYPEs to include the dimensionality of the data. This can be accomplished with a simple update statement with a commit.

In the case above, the SDO_GTYPE column is being altered for all rows in the table to specify that the data is 2-dimensional. If the SDO_GTYPE was 1 (for point data), then it would be altered to 2001 (for two-dimensional point data).

Oracle Spatial does not support partially migrated data. If data has been changed to have the four digit SDO_GTYPE, but the polygon element types have not been changed, or the rotation rules and/or order rules for polygons have not been adhered to, then the data will be considered invalid.

Any updates of data in the SDO_GEOMETRY object cause the spatial index to be updated. For this reason, Oracle recommends that whenever migrating data, drop the spatial index before the migration, migrate the data, and then recreate the index after migration.

Summary

In this lesson, you should have learned how to:

- **Identify the methods for loading spatial data**
- **Create the control files associated with the bulk loading of spatial data**
- **Use the shapefile converter tool to convert an ESRI shapefile into a SQL*Loader control file**
- **Verify the validity of individual geometries, or of an entire layer**
- **Use the migration tool to convert object-relational data from Oracle Spatial 8.1.5 to Oracle Spatial 8.1.6 and higher**

ORACLE

6

Indexing Spatial Data

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Explain the concept of spatial indexing**
- **Generate a quadtree spatial index**
- **Generate an R-tree spatial index**

ORACLE

6-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

This lesson describes the spatial indexing capabilities of Oracle Spatial. It discusses the different types of spatial indexes and how to create them.

Spatial Indexing

Two methods for indexing spatial data:

- **Quadtree indexing**
 - Successive decomposition of space into tiles
 - Two types of quadtree indexes:
 - Fixed
 - Hybrid (not recommended)
- **R-tree indexing**
 - Based on minimum bounding rectangles

ORACLE

6-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Indexing Geometric Features Using a Spatial Index

Once spatial data has been loaded into Oracle tables through either bulk or transactional loading, a spatial index may need to be created. Spatial indexes are built for the same reason other indexes are built in a database: for fast access to data. In the case of a spatial index, the index is built on the spatial column (layer) in the table to provide fast access to data in a particular region.

Oracle Spatial provides two distinct methods for indexing spatial data.

- Quadtree Indexing is accomplished by successively decomposing the coordinate space into tiles. The process of determining which tiles cover a given geometry is called “tessellation.” Spatial indexing routines use the tiles that cover a geometry as an approximation of that geometry. There are two types of quadtree indexes:
 - Fixed indexing, which uses fixed size tiles to approximate each geometry
 - Hybrid indexing, which uses a combination of fixed and variable sized tiles to approximate each geometry. Hybrid indexing is not recommended.
- R-tree indexing is accomplished by building a Minimum Bounding Rectangle (MBR) around each geometry. Spatial indexing routines then use the MBR as an approximation of the geometry.

Spatial indexes store approximations of geometries as either one or more tiles (quadtree) or as a minimum bounding rectangle (R-tree). Because they compare approximations of geometries rather than the geometries themselves, spatial indexes can work quickly on large amounts of spatial data. This will be explained in more detail throughout this lesson.

Quadtree Indexing

Based on the successive decomposition of a coordinate space by factors of four.

- **This decomposition is called tessellation.**
- **Each geometry is mapped to the tessellated coordinate space, and each tile that covers any part of the geometry is stored in the spatial index.**
- **The tessellation process is performed by creating the spatial index.**

ORACLE

6-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Quadtree Indexing

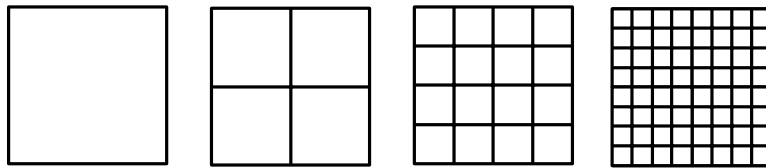
Quadtree indexing is based on the successive decomposition of a coordinate space by factors of four. This means that the coordinate space is first subdivided into four parts (or tiles), then each of those four tiles are subdivided into four tiles (resulting in a total of 16 tiles), then each of those tiles is subdivided into four tiles (resulting in 64 tiles), and so on until a level of tiling is accomplished as specified by the user.

When a spatial index is created, the coordinate system space is decomposed, and the process of determining which tiles cover a given geometry is completed. For each geometry, any tile that covers it is stored in the spatial index. Each geometry can be covered by one or more tiles, so each geometry can have one or more spatial index entries. Only tiles that cover geometries are stored in the spatial index.

Building a quadtree spatial index is a geometry-by-geometry operation, that is, the first geometry in the table is tessellated, then the next geometry, then the one after that, and so on until all geometries in the table have been indexed. This is in contrast to creating an R-tree spatial index, which creates the index for the entire spatial layer all at once.

Tessellation

Fixed Tile Decomposition



ORACLE

6-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Tessellation Example

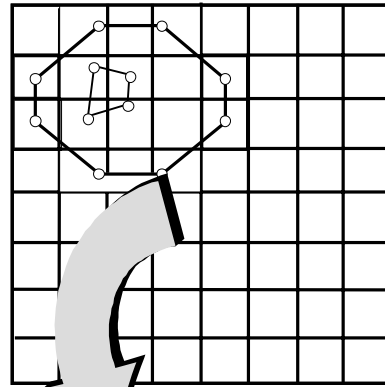
Note that you start with a single tile the size of the entire coordinate system. The concept of tiling level will now be introduced. Tiling level directly corresponds to the size of a tile.

- At level 1 (or the first level of decomposition), the coordinate system is decomposed into 4 tiles.
- At level 2, the coordinate system is decomposed into 16 tiles.
- At level 3 the coordinate system is decomposed into 64 tiles.
- At each level there are 4^L possible tiles, where L is the level.

Note that the size of the tiles decreases very quickly as the level of decomposition increases. Given coordinate system bounds of -180 to 180 in longitude, and -90 to 90 in latitude, the size of a tile at the Earth's equator at level 32 is approximately 9mm by 4mm.

It is important to remember that tiles are only generated where there is data. In the “Tuning and Administration” lesson you will cover how to pick an optimal tiling level (or tile size) for a quadtree spatial index.

How a Geometry Is Indexed Using Quadtree Indexing



- DBA Determines
– Level = 3

| SDO_CODE |
|--------------|
| FIXED_TILE1 |
| FIXED_TILE2 |
| ... |
| FIXED_TILE16 |

ORACLE

6-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Indexing a Geometry by Using Quadtree Indexing

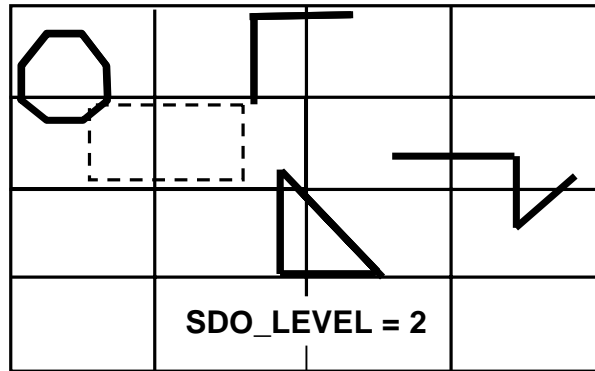
In the above example, a DBA has determined that a Level 3 tessellation is appropriate for this geometry. The Level 3 is useful for illustrative purposes only. A Level 3 tessellation is almost never appropriate when tiling a spatial layer.

In this example, 16 Level 3 tiles are generated for the geometry. The tiles that cover any part of the geometry being indexed are stored in the SDO_CODE column of the spatial index table.

Note that it is not just tiles that cover the boundary of the geometry which are stored, it is all tiles that cover any part of the geometry (boundary or interior).

Other items stored in the spatial index table include a ROWID pointer back to the geometry being indexed, and a status field that holds information about whether each tile covers a boundary portion of the geometry or an interior portion of the geometry.

Primary Filter Example (Quadtree Index)



- **Compares fixed sized tiles that approximate the area of interest with fixed sized tiles that approximate each geometry**
- **Result is not exact because comparing approximations**

Primary Filter Example Using a Quadtree Index

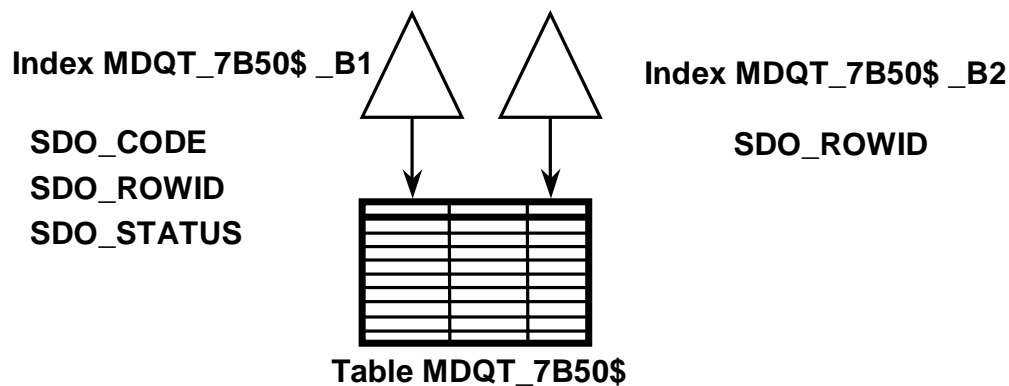
In this example, the index is being used to determine which geometries have any interaction with the rectangular geometry whose boundary is represented by the dotted line.

The quadtree spatial index is used to store approximations of each of the geometries shown. Every tile that covers part of each geometry is stored in the spatial index along with a pointer to that geometry. The query window (the rectangle whose boundary is a dotted line) is also spatially indexed. There are two tiles associated with the dotted rectangle. The spatial index works by comparing approximations of geometries. In this quadtree index case, the approximations are tiles. All geometries that have an index entry that includes either of the two tiles that approximate the rectangle are returned by the spatial index query. In this example, three geometries are returned by the spatial index.

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 (or primary filter) will always return the correct final results of a spatial query, and perhaps some extra geometries as well.

A Look at Quadtree Index Structures

```
create index STATES_SIDX on STATES (GEOM)
  indextype is MDSYS.SPATIAL_INDEX
  parameters ('SDO_LEVEL=8');
```



ORACLE

6-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Quadtree Index Structures

When a quadtree spatial index is built, an Oracle table is created. This table (also known as the spatial index table) has three columns:

- **SDO_CODE:** This is the tile code for the tile that the geometry is associated with. Each geometry must be associated with at least one tile. Single point geometries are guaranteed to be associated with exactly one tile. Other types of geometries may have many tiles associated with each geometry.
- **SDO_ROWID:** This is a pointer back to the ROWID of the geometry the tile is associated with. Because each geometry can be associated with more than one tile, the ROWID of each geometry can appear in the spatial index table more than one time. Again, point geometries are an exception to this. The ROWID of point geometries will only appear once in the spatial index table.
- **SDO_STATUS:** This is a marker for whether the tile is an interior tile or a boundary tile for the geometry it is associated with. This will be discussed further in the “Tuning and Administration” lesson.

Quadtree Index Structures (continued)

After the spatial index table is created and populated, two Oracle B-tree indexes are built on the spatial index table. The first index is built on all three columns of the spatial index table. The second index is built only on the SDO_ROWID. These indexes provide fast lookup into the spatial index table.

The internal structures (the spatial index table and the two B-tree indexes) are visible to the end user. They are flagged in Oracle's metadata tables (xxx_TABLES and xxx_INDEXES) as SECONDARY = 'Y'. The spatial index name will always appear to be in the SYSTEM tablespace (if it is looked up in USER_INDEXES). This is because the spatial index name is only a logical structure associated with three physical structures (the spatial index table and two B-tree indexes on the spatial index table). The tablespace and storage parameters can be specified for the physical structures associated with the spatial index name. This will be demonstrated later when the CREATE INDEX command is discussed.

What Happens During Quadtree Index Creation

- **Create the index table**
- **Populate the index table**
 - Read the geometries, perform the tessellation and insert into the index table
- **Create the B-tree index on tile codes, rowids, and status**
- **Create the B-tree index on rowids**

ORACLE

6-10

Copyright © Oracle Corporation, 2001. All rights reserved.

What Happens During Quadtree Index Creation

When a quadtree spatial index is created, Oracle Spatial first creates the spatial index table that will be used to hold the spatial index information. The spatial index table has three columns: SDO_CODE, SDO_ROWID, and SDO_STATUS.

Next, the spatial index table is populated. To populate the spatial index table, each geometry is read, tessellated (or tiled), and the results of the tessellation are inserted into the spatial index table.

When the tessellation process is completed for all geometries, an Oracle index is built on the SDO_CODE, SDO_ROWID, and SDO_STATUS fields of the spatial index table, and another Oracle index is built on the SDO_ROWID column.

When all of the above has completed, the creation of the quadtree spatial index is complete.

How Much of my Quadtree Index Has Been Built?

- Can monitor if using SDO_COMMIT_INTERVAL
- Use plain SQL queries on the spatial index table:
 - First get the name of the index table
 - Then get the count of the geometries that have been indexed

```
SQL> select sdo_index_table
  2   from user_sdo_index_metadata
  3   where sdo_index_name = 'STATES_SIDX';

SDO_INDEX_TABLE
-----
MDQT_7B50$

SQL> select count (distinct (sdo_rowid))
  2   from MDQT_7B50$;
```

ORACLE

6-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Determining Quadtree Index Build Progress

It is possible to determine how far a quadtree index build has progressed by using simple SQL statements.

Building a quadtree spatial index is a geometry-by-geometry operation. If the spatial index parameter SDO_COMMIT_INTERVAL has been specified, then the results of the quadtree index build are committed every time the number of geometries specified has been indexed.

To determine how many geometries have been indexed, first determine the name of the spatial index table by querying the USER_SDO_INDEX_METADATA table. Once the name of the table has been determined, then query to determine the distinct values in the SDO_ROWID column in the spatial index table. This will be the number of geometries that have been indexed so far in the spatial index table. This number can be compared with the total number of geometries in the spatial layer.

Note that after all of the geometries have been spatially indexed, the two B-tree indexes are then built on the spatial index table. Although all of the geometries may appear in the spatial index table, the index build is not complete until the B-tree indexes have been built.

R-tree Indexing

- **Another common way to index spatial data**
- **Requires less tuning**
- **Extends primary filter to 3 or 4 dimensions**
- **Each index entry approximates geometry using MBR (Minimum Bounding Rectangle) for 2D, and MBV (Minimum Bounding Volume) for 3D**
- **MBRs/MBVs indexed internally using a tree structure**

ORACLE

6-12

Copyright © Oracle Corporation, 2001. All rights reserved.

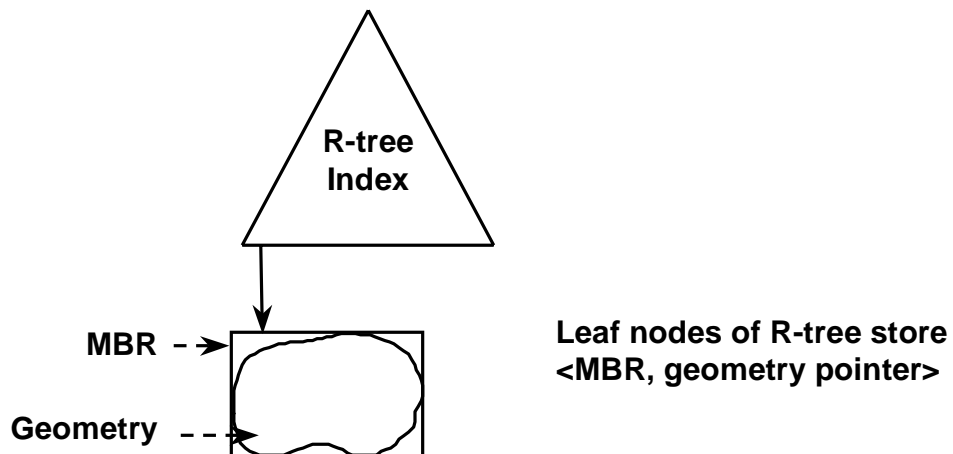
R-tree Indexing

R-tree indexing is another common way to index spatial data. R-tree indexes are very simple to create, and there is very little/no tuning required to achieve optimal performance. While quadtree indexes can only index two-dimensional data, R-tree indexing can be done on two, three, or four dimensions.

R-tree indexes are based on minimum bounding rectangles for two-dimensional data. This means that a rectangle is built on each geometry based on the minimum values for each dimension and the maximum values for each dimension. When indexing three-dimensional 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 data is sorted and an index tree structure is built and written to a table. In the case of R-tree indexes, the table structure is all that is created. The table structure is the index.

R-tree Indexing Concept



ORACLE

6-13

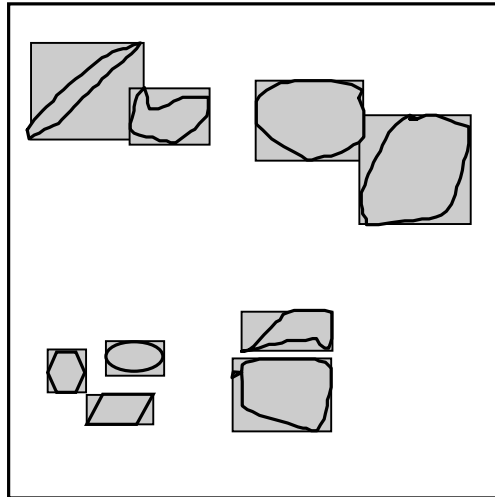
Copyright © Oracle Corporation, 2001. All rights reserved.

R-tree Indexing Concept

Starting with a geometry, the R-tree will build a minimum bounding rectangle around that 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. Note that unlike quadtree indexes, R-tree indexes by definition will get only one index entry for each geometry.

How Geometries Are Indexed Using R-trees



ORACLE

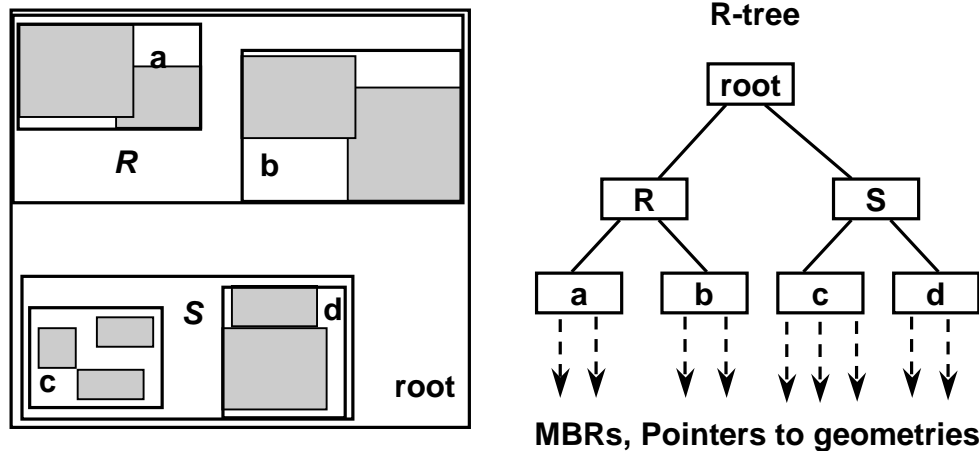
6-14

Copyright © Oracle Corporation, 2001. All rights reserved.

How Geometries Are Indexed Using R-trees

When an R-tree index is built, a Minimum Bounding Rectangle (MBR) is built around each geometry. In this example, there are nine minimum bounding rectangles created, one for each geometry.

How Geometries Are Indexed Using R-trees



ORACLE

6-15

Copyright © Oracle Corporation, 2001. All rights reserved.

How Geometries Are Indexed Using R-trees (continued)

After the MBRs are built around each geometry, the data is 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, we see that **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**.

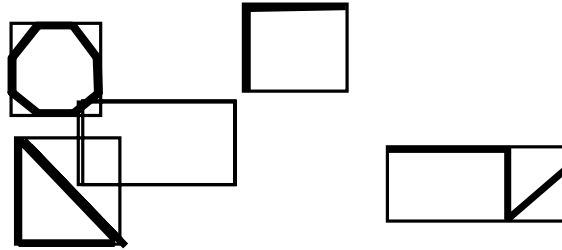
The top level (or *root*) of the R-tree index, contains pointers to **R** and **S** as well as the minimum bounding rectangles associated with both of those nodes.

Note that the R-tree structure is optimized for accessing elements that are in close proximity to each other.

Also, note that an R-tree is always accessed from the root down; there are no backwards pointers in the R-tree index structure.

The R-tree information is presented in this way for simplicity. In Oracle Spatial, the default fanout for the R-tree is 35, meaning that the *root* node would contain pointers to 35 minimum bounding rectangles, and each of the 35 minimum bounding rectangles would have pointers to 35 other minimum bounding rectangles, etc. until the leaf nodes are reached, which would have pointers to 35 geometries and their MBRs. In this way many geometries can be indexed without the R-tree being too deep.

Primary Filter Example (R-tree Index)



- **Compares MBR that approximates the area of interest with the MBR that approximates each geometry**
- **Result is not exact because comparing approximations**

Primary Filter Example Using an R-tree Index

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

The R-tree spatial index is used to store approximations of each of the geometries shown. Every minimum bounding rectangle that covers each geometry is stored in the spatial index along with a pointer to that geometry. The query window (the rectangle) is also spatially indexed. The spatial index works by comparing approximations of geometries. In this R-tree index case, the approximations are minimum bounding rectangles. All geometries that have an index entry whose minimum bounding rectangle interacts with the minimum bounding rectangle that approximates the query window are returned by the spatial index query. In this example, two geometries are returned by the spatial index.

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 (or primary filter) will always return the correct final results of a spatial query, and perhaps some extra geometries as well.

Quadtrees or R-trees?

Benchmarking in progress to determine which index is best under certain conditions

- **Density of data?**
- **Uniform distribution?**
- **Size of geometries?**
- **Alignment of geometries?**
- **Shape of geometries?**
- **Types of queries?**

ORACLE

6-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Which Spatial Index Type to Use?

Work is currently under way in Oracle Spatial engineering to help determine under which conditions use of quadtree versus R-tree indexes is appropriate. Some items under study include:

- **Density of data:** If the data is very sparse then the choice of spatial index type becomes less important
- **Distribution of data:** Depending on where typical query windows are with respect to the distribution of data, choosing an index type may or may not be important. Again, if the typical query window is in a sparse region with respect to spatial data, then the choice of index type may be less important
- **Size of geometries:** If the geometries are large in comparison to a typical query window, then the choice of index type becomes less important. If the geometries are small in comparison to the query window, and the data is dense, then the choice of spatial index type becomes more important.

Which Spatial Index Type to Use? (continued)

- Alignment of geometries: How the geometries are aligned with respect to the x and y axes may have an effect on the efficiency of the index. Remembering that an index is an approximation of a geometry, a rectangle aligned on the x and y axis can be perfectly approximated by a minimum bounding rectangle using an R-tree index. If the same geometry is offset from the x and y axis (that is, tilted on an angle) then the approximation of that rectangle could now include a lot of space in which no part of the geometry exists.
- Shape of geometries: If the geometries are very regular in shape (for instance, rectangles) then R-tree indexes may be more effective. Again, the index that best approximates the geometries may be the best index to use in general. However, because of other optimizations within Oracle Spatial, there may be situations where the difference in the approximations can be overcome.
- Types of queries: R-tree indexes have shown to be particularly effective for doing the nearest neighbor type of queries.

Quadtrees or R-trees?

- **R-tree Strengths**

- Easier to create
- Use less storage
- Faster for nearest neighbor
- Extend primary filter to three or four dimensions
- Can be geodetic
- Support incremental nearest neighbor

- **R-tree Weaknesses**

- Can be slower when updating/adding/deleting data
- May become less efficient with frequent updates (can rebuild)
- May not be appropriate for all data sets

ORACLE

6-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Quadtrees or R-trees?

R-tree indexes have strengths and weaknesses. Strengths of R-tree indexes include that they:

- Are easier to create: There is no specialized tuning knowledge required to use R-tree indexes
- Are faster for nearest neighbor: Because the R-tree index stores spatial data in the index by location, this is the optimal index to use for nearest neighbor types of queries
- Use less storage: If data storage requirements need to be minimized, then R-tree indexes are best
- Extend the primary filter to three or four dimensions: R-trees can index two-, three-, or four- dimensional spatial data. If you need to index more than two dimensions, then R-trees are the only choice of index type.
- Can be geodetic: if you need full geodetic support for spatial operators (spatial queries that use the spatial index, covered in a later lesson), then R-tree indexes are the only index type that has complete geodetic support.
- Support incremental nearest neighbor: Nearest neighbor functionality will be covered in a later lesson. R-tree indexes are the only index type that has support for incremental nearest neighbor, where results from nearest neighbor keep getting returned until some nonspatial criteria are met.

Quadtrees or R-trees? (continued)

Some of the weaknesses of R-trees include that they:

- May become less efficient when lots of updates (can rebuild): R-tree indexes can become less efficient over time if there is a lot of data added over time. The index can be rebuilt to restore optimal efficiency.
- Are slower when updating/adding/deleting data: Index maintenance takes longer with R-trees than it does with quadtrees.
- May not be appropriate for every data set.

Quadrees or R-trees?

- **Quadtree Strengths**
 - Maintain their efficiency even after lots of updates
 - Can be faster when updating, inserting, or deleting data
 - Work well independent of geometry type or shape
- **Quadtree Weaknesses**
 - Require tuning
 - Are exclusively two-dimensional
 - Cannot be geodetic
 - Do not support incremental nearest neighbor

ORACLE

6-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Quadrees or R-trees (continued)?

Some of the strengths of quadtree indexes include that they:

- Maintain their efficiency after lots of updates: Quadtree indexes are not affected by large amounts of updates in the same way R-trees are.
- Can be faster when updating, inserting, or deleting data: If insert, update, or delete performance of spatial data is a major concern, quadtree indexes may be a better choice.
- Work well for nearly every kind of data set.

Some of the weaknesses of quadtree indexes include that they:

- Require tuning: specialized knowledge of tessellation and tiling are required to build an effective quadtree spatial index.
- Are two-dimensional: Quadtree indexes are always two-dimensional.
- Cannot be geodetic: Geodetic indexes include a third dimension in the index. Quadtree indexes cannot be geodetic.
- Do not support incremental nearest neighbor: If incremental nearest neighbor is required, then quadtree indexes should not be used. If high performance for nearest neighbor queries are required, then quadtree indexes should not be used.

CREATE INDEX

```
CREATE INDEX <index-name>
ON <table-name> (<column-name>)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
[PARAMETERS (
  'SDO_LEVEL      = <level>,
  SDO_COMMIT_INTERVAL = <n>,
  [<storage_parameters> = <param_value> ... '
)];
```

ORACLE

6-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax

As mentioned previously, if you know how to do operations in Oracle, then you know most of what you need to know to use Oracle Spatial. The CREATE INDEX statement above is exactly the same as any Oracle CREATE INDEX statement, with some additional information added in.

The new required syntax is INDEXTYPE IS MDSYS.SPATIAL_INDEX.

Optionally, other parameters can be specified as well. The syntax is discussed more fully in the next slides.

Complete syntax information for creating Oracle Spatial indexes can be found in the *Oracle Spatial User's Guide and Reference*.

CREATE INDEX

- **<index-name>**
 - Name of index
- **<table-name>**
 - Table that contains the column of type SDO_GEOMETRY
- **<column-name>**
 - Name of the column of type SDO_GEOMETRY

ORACLE

6-23

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax (continued)

All three of the syntax elements on this page are the same as those used for any other Oracle CREATE INDEX statement.

All indexes in Oracle must be named. The <index-name> specifies the name of the index. Normal Oracle index naming conventions apply to the index name. The length of the name has to be 30 characters or less, and the index name must start with an alphabetic character.

The <table-name> is the name of the table that contains the spatial column to be indexed. The spatial column is of type MDSYS.SDO_GEOMETRY.

The <column-name> is the name of the column of type MDSYS.SDO_GEOMETRY that is to be spatially indexed.

CREATE INDEX (Quadtree)

SDO_LEVEL

- Required when building quadtree indexes
- This number defines the fixed tile level portion of the spatial index
- The level determines the size of the fixed sized tile
- SDO_LEVEL=1 has tiles the size of 1/4 the coordinate system, SDO_LEVEL=2 has tiles the size of 1/16 the coordinate system, and so on
- If not specified, an R-tree index is built

ORACLE

6-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax (Quadtree)

The following parameters of CREATE INDEX are optional when creating a spatial index. They follow the PARAMETERS keyword in the create index statement, and appear inside a quoted string inside parentheses.

SDO_LEVEL is used to build quadtree indexes.

The number chosen for SDO_LEVEL defines the quadtree tile size. Choosing a higher number means a smaller tile size. For every increment of SDO_LEVEL, the tile size becomes 1/4 the tile size at the previous SDO_LEVEL. For instance, at SDO_LEVEL=1, each tile is 1/4 the size of the coordinate system. At SDO_LEVEL=2, each tile is 1/4 the size of SDO_LEVEL=1 sized tiles. At SDO_LEVEL=3, each tile is 1/4 the size of SDO_LEVEL=2 sized tile, etc.

If SDO_LEVEL is not specified, or is 0 (zero) then an R-tree index is built.

CREATE INDEX (Quadtree)

SDO_COMMIT_INTERVAL

- **Commit after this many geometries are indexed**
- **Default value is -1. By default commit is done at the end of the entire transaction**

ORACLE

6-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax (Quadtree) (continued)

Additional quadtree index-specific syntax includes:

- **SDO_COMMIT_INTERVAL**: Set **SDO_COMMIT_INTERVAL=N** where the indexing results are committed to the spatial index table every N geometries. This is useful to monitor how many geometries have been indexed during the spatial index creation process (see the discussion earlier in this lesson). The default value for **SDO_COMMIT_INTERVAL** is -1, which means to commit the results of the spatial indexing after all of the geometries have been indexed. Note that spatial indexing with quadtrees is a geometry-by-geometry operation until all the geometries have been indexed, then B-tree indexes are created on the spatial table.
- **SDO_COMMIT_INTERVAL** is not used for R-tree indexes because R-tree index creation is not a geometry-by-geometry operation. When an R-tree index is created, the entire index structure is created as a result of processing all geometries in the table.

CREATE INDEX (Quadtree: Hybrid Indexes Only)

- SDO_NUMTILES
- SDO_MAXLEVEL

These parameters are hybrid index parameters. Use of hybrid indexes is not recommended in Oracle Spatial. Do not specify these parameters.

ORACLE

6-26

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax (Quadtree) (continued)

Note on hybrid indexes:

In nearly all spatial data sets, optimal performance can be achieved using either fixed quadtree or R-tree indexes. Oracle does not recommend using hybrid indexes. This topic has been included for completeness in the discussion of spatial indexing.

So far the only type of quadtree index discussed has been a fixed sized tile quadtree index. There is an additional quadtree index type called a *hybrid* quadtree index. A hybrid quadtree index uses fixed sized tiles and additional variable sized tiles to approximate geometries. The user specifies the variable sized tile component using the parameter SDO_NUMTILES. SDO_NUMTILES specifies the minimum number of tiles (and thus index entries) that must be generated for each geometry before that geometry can be considered indexed. For instance, if at a given SDO_LEVEL there were three tiles generated for a geometry, but the user had specified SDO_NUMTILES=6, then further decomposition of the coordinate space would take place for that geometry such that more smaller tiles would cover it.

Create Index Syntax (Quadtree) (continued)

SDO_NUMTILES defines how many variable sized tiles to generate for a geometry. Variable sized tiles will always be equal to or smaller than fixed sized tiles (the SDO_LEVEL defines the largest size tile to be used). Because point data always has only one index entry, specify SDO_NUMTILES=1 for point data. If nonpoint data, set SDO_NUMTILES=4 or higher. The OEM tool *Spatial Index Advisor* had a bug in 8.1.6 (fixed in 8.1.7) such that the only way to create a fixed index was to set SDO_NUMTILES=0.

SDO_MAXLEVEL defines the maximum level for any variable sized tile. Tiles will never be smaller than the tile size specified by SDO_MAXLEVEL. The default value for SDO_MAXLEVEL is 32. Oracle recommends that this value should not be changed.

CREATE INDEX (R-tree)

- **SDO_INDX_DIMS**
 - Defines the number of dimensions on which the index is built
 - Useful to index more than two dimensions
 - Default is two (was number of dimensions in 8.1.7)
- **SDO_RTR_PCTFREE**
 - Percentage of slots left empty in index nodes to accommodate future insertions
 - Default is 10 percent, that is, 10% of nodes in the tree are empty nodes

ORACLE

6-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax (R-tree)

The following parameters are specific to R-tree index creation only:

- **SDO_INDX_DIMS** specifies the number of dimensions to spatially index. For instance, if the spatial data has three dimensions but the index should only be built on two dimensions, then **SDO_INDX_DIMS=2** should be specified. This is common when using linear referencing. In linear referencing, the first two dimensions may be x and y and the next dimension may be the measure value. The default number of dimensions to index is the number of dimensions that appear in **USER_SDO_GEOM_METADATA**. An example of when **SDO_INDX_DIMS** should be used to override the default is when the spatial data is linear referenced. The index should not be built on the measure value. **SDO_INDX_DIMS** can restrict the index to the nonmeasure portion of the data.
- **SDO_RTR_PCTFREE** specifies the number of slots to leave empty in the R-tree to accommodate future insertions/updates. As mentioned previously, 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 of reducing 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% more data over time, specifying **SDO_RTR_PCTFREE=50** would leave room in the index for that data. The default value (if **SDO_RTR_PCTFREE** is not specified) is 10 percent, that is, 10 percent of the nodes are left empty to accommodate future data.

CREATE INDEX

LAYER_GTYPE

- **Layer type constraint for all spatial index types**
 - Will only allow geometries of given type to be indexed
 - 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

6-29

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax

The following syntax is useful for both quadtree and R-tree indexes. It is new syntax for Oracle9i.

LAYER_GTYPE is 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 to having geometry type constraints. First, to prevent data from being inserted into a spatial layer if it is not of a predefined type. Second, to make use of type 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 the MULTIPOINT LAYER_GTYPE). 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

GEODETIC

- **Set to FALSE to build a Cartesian index on geodetic data (data with a geodetic SRID)**
 - Only way to create a quadtree index on geodetic SRIDs
 - Can specify to create a non-geodetic R-tree index
 - Some key functionality is disabled for the spatial operators SDO_NN and SDO_WITHIN_DISTANCE (for example, the use of UNIT)
- **Default is TRUE**
- **Not recommended to change the default value**

ORACLE

6-30

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax

Additional generic syntax:

GEODETIC: This is new syntax for Oracle9i. Set GEODETIC=FALSE to build a Cartesian index on spatial data with a geodetic SRID.

Setting GEODETIC=FALSE provides backwards compatibility for existing applications that require quadtree indexes on geodetic data. If an attempt is made to build a quadtree index on geodetic data without this parameter an Oracle error is raised.

This parameter can also be used to build a nongeodetic R-tree index.

The default for Oracle Spatial is GEODETIC=TRUE.

When GEODETIC=FALSE is specified, then the UNITS parameter is disabled for the spatial operators SDO_NN and SDO_WITHIN_DISTANCE.

CREATE INDEX

<storage_parameters>

- **Storage parameters for the index table**
- **initial, next, pctincrease, minextents, maxextents, and tablespace**
- **For quadtree indexes, can define B-tree index storage parameters on spatial index table (btree_initial, btree_next, btree_pctincrease)**

ORACLE

6-31

Copyright © Oracle Corporation, 2001. All rights reserved.

Create Index Syntax (continued)

Storage parameters can be specified in the parameters string to provide information to Oracle regarding storage of the spatial index table.

The following can be specified:

INITIAL: The size of the initial extent of the spatial index table

NEXT: The size of the next extent of the spatial index table

PCTINCREASE: The percentage to increase the previously used extent by when the need to create a new extent arises

MINEXTENTS: The minimum number of extents to create for the spatial index table

MAXEXTENTS: The maximum number of extents to create for the spatial index table

TABLESPACE: The name of the tablespace to put the spatial index table in

Additionally, for quadtree indexes only, the following can be specified:

BTREE_INITIAL: The size of the initial extent of the B-tree index on the spatial index table

BTREE_NEXT: The size of the next extent of the B-tree index on the spatial index table

BTREE_PCTINCREASE: The percentage to increase the previously used extent by when the need to create a new extent for the B-tree index arises

DROP INDEX

```
DROP INDEX <index_name>;
```

- Drops all index tables associated with a column of type **SDO_GEOMETRY**.
- If a **CREATE INDEX** on a column of type **SDO_GEOMETRY** does not successfully complete, first try **DROP INDEX**, and if that does not work, you must call **DROP INDEX** using the **FORCE** option.

```
DROP INDEX <index_name> FORCE;
```

ORACLE

6-32

Copyright © Oracle Corporation, 2001. All rights reserved.

DROP INDEX Syntax

Spatial indexes created with the **CREATE INDEX** syntax can be dropped using the **DROP INDEX** syntax. Dropping a spatial index drops all of the spatial index tables and indexes associated with those tables based on the name of the index being dropped.

Sometimes a **CREATE INDEX** on a column of type **SDO_GEOMETRY** does not successfully complete. If **DROP INDEX index_name** does not work after a spatial index build is unsuccessful, the **FORCE** option should be used in the **DROP INDEX** statement, that is, **DROP INDEX index_name FORCE**;

Complete syntax information for dropping Oracle Spatial indexes can be found in the *Oracle Spatial User's Guide and Reference*.

ALTER INDEX

```
ALTER INDEX <index_name>  
PARAMETERS (<parameter string>);
```

```
ALTER INDEX <index_name>  
REBUILD  
PARAMETERS (<parameter string>);
```

```
ALTER INDEX <index_name>  
RENAME TO  
<new_index_name>;
```

ORACLE

6-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Alter Index Syntax

ALTER INDEX syntax can be used with Oracle Spatial indexes.

Oracle columns, spatial or nonspatial, can only have one index, but spatial columns can have multiple spatial index tables. This will be discussed in detail in the next slide.

ALTER INDEX with parameters allows spatial index tables to be added or deleted from a spatial index.

ALTER INDEX with the REBUILD option allows a spatial index table to be rebuilt, possibly with different parameters than were specified the previous time the spatial index table was built.

ALTER INDEX with the RENAME TO clause renames the index.

Complete syntax information for altering Oracle Spatial indexes can be found in the *Oracle Spatial User's Guide and Reference*.

ALTER INDEX

```
ALTER INDEX <index_name>  
PARAMETERS (<parameter string>);
```

- An SDO_GEOMETRY column's spatial index may have multiple “spatial index tables”
- Parameters are passed in as a quoted string
 - 'add_index=<index_table_name> sdo_level=<level>'
 - 'delete_index=<index_table_name>'
- Cannot delete the primary “spatial index table” (use DROP INDEX)

ORACLE

6-34

Copyright © Oracle Corporation, 2001. All rights reserved.

ALTER INDEX

This ALTER INDEX syntax is used to add additional “spatial index tables” to an existing spatial index. For instance, if there is an R-tree index on a spatial column, and an additional quadtree index is desired, then the add_index parameter can be passed in along with any quadtree index parameters, for example:

```
ALTER INDEX MY_SIDX PARAMETERS ('add_index=my_sidx_qtree_tab  
sdo_level=12');
```

Similarly, an index that was added using add_index can be deleted using delete_index:

```
ALTER INDEX MY_SIDX PARAMETERS ('delete_index=my_sidx_qtree_tab');
```

A primary spatial index table (the spatial index table created by the CREATE INDEX command) cannot be deleted using using this mechanism. It must be dropped using DROP INDEX syntax.

ALTER INDEX

```
ALTER INDEX <index_name>
REBUILD
PARAMETERS (<parameter string>);
```

- Rebuild a spatial index table. You can change the tiling level of a spatial index table. This will rebuild the entire spatial index table.
- Parameters are passed in as a quoted string.
 - *'rebuild_index=<index_table_name>
sdo_level=<level>'*

ORACLE

6-35

Copyright © Oracle Corporation, 2001. All rights reserved.

ALTER INDEX REBUILD Syntax

This command causes the entire spatial index to be rebuilt. All parameters associated with the index type can be changed in the parameter list of the ALTER INDEX REBUILD. For instance, storage parameters, SDO_LEVEL, SDO_COMMIT_INTERVAL, and so on can all be specified for an existing quadtree index. R-tree indexing parameters can also be re-specified during a rebuild. LAYER_GTYPE can be re-specified also, which causes the entire index to be rebuilt.

If a quadtree index is rebuilt, specifying the SDO_COMMIT_INTERVAL will allow users to monitor the progress of the rebuild the same way it does for CREATE INDEX.

ALTER INDEX

```
ALTER INDEX <index_name>  
RENAME TO  
<new_index_name>;
```

- **Change the name of the spatial index.**
- **Can only rename the index, not the index tables.**

ORACLE

6-36

Copyright © Oracle Corporation, 2001. All rights reserved.

ALTER INDEX RENAME

This syntax causes the spatial index to be renamed. It only renames the spatial index, not the objects associated with the spatial index (such as the index table, or the B-tree indexes on a spatial quadtree index table).

Spatial Index Dictionary Views

Similar to other Oracle dictionary views

- **USER_SDO_INDEX_METADATA**
- **ALL_SDO_INDEX_METADATA**
- **DBA_SDO_INDEX_METADATA**

ORACLE

6-37

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Dictionary Views

Three views are defined that contain the metadata associated with spatial indexes. They are `USER_SDO_INDEX_METADATA`, `ALL_SDO_INDEX_METADATA`, and `DBA_SDO_INDEX_METADATA`. They contain all of the index metadata associated with spatial indexes. Unlike the `*_SDO_GEOM_METADATA` views where the user is responsible for populating and maintaining their view, the `*_SDO_INDEX_METADATA` views are entirely maintained by Oracle Spatial.

The `USER_SDO_INDEX_METADATA` view shows all of the spatial index metadata for the current user, the `ALL_SDO_INDEX_METADATA` view shows all of the spatial index metadata the current user has select privilege for, and the `DBA_SDO_INDEX_METADATA` view shows all of the spatial index metadata for the instance to privileged DBA users.

USER_SDO_INDEX_METADATA (Spatial Index Dictionary 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_RTREE_SEQ_NAME | VARCHAR2(32) |
| SDO_RTREE_PCTFREE | NUMBER |
| 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) |

ORACLE

6-38

Copyright © Oracle Corporation, 2001. All rights reserved.

USER_SDO_INDEX_METADATA View

Above are the columns associated with the USER_SDO_INDEX_METADATA view. Information about each column is available in the *Oracle Spatial User's Guide and Reference*.

Spatial Index Informational Views

Combines dictionary views

- **USER_SDO_INDEX_INFO**
- **ALL_SDO_INDEX_INFO**
- **DBA_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

6-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Informational Views

Oracle Spatial includes a view that combines the system views *_INDEXES (USER, ALL, DBA) with the Oracle Spatial specific views *_SDO_INDEX_METADATA. The two views are joined based on the index name to provide an easy mechanism to see important index information without requiring end users or applications to write complicated SQL.

Summary

In this lesson, you should have learned how to:

- **Explain the concept of spatial indexing**
- **Explain the procedures associated with spatial indexing**
- **Generate a quadtree spatial index**
- **Generate an R-tree spatial index**
- **Describe some of the factors affecting performance of R-tree and quadtree indexes**

ORACLE



Tuning and Administration

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Determine the appropriate tiling level when building a quadtree index on a spatial layer**
- **Use the OEM tool Spatial Index Advisor to create and tune spatial indexes**

ORACLE

7-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

This lesson describes the process used to determine an optimal tiling level for a layer when building quadtree spatial indexes.

Additionally, this lesson shows how Spatial Index Advisor can be used to index data and look at the quadtree index structures.

This lesson will also describe how to verify the validity of spatial data, both on a geometry-by-geometry basis and as a layer operation.

Tuning and Administration Tools

- **The SDO_TUNE package**
 - Spatial data analysis
 - Tune spatial indexes
- **Enterprise Manager**
 - Spatial Index Advisor
- **The SDO_GEOM package**
 - Validating geometries

ORACLE

7-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Tuning and Administration Tools

There are several tools available to help tune and administer spatial data.

The functions that will be described in the SDO_TUNE package are used to understand more about the data that will be indexed, as well as to provide insight as to what a reasonable starting point for SDO_LEVEL might be when creating a quadtree spatial index.

Another tool available to tune and administer spatial data is the Spatial Index Advisor tool within Oracle Enterprise Manager. This tool provides a number of useful functions to help understand the relationship between the data in the spatial layer and the quadtree index that is built on it. It also helps users determine the best tiling level with which to start index analysis, and will create the spatial index as well. The Spatial Index Advisor uses functionality described in the SDO_TUNE package, and additionally does more sophisticated analysis.

Additional tools from the SDO_GEOM package are described that help users determine the validity of their geometries either on a single geometry at a time or at the layer level.

What Type of Index Should I Use?

- **Fixed quadtree indexes are very good in nearly all cases, but require tuning knowledge.**
- **R-tree indexes are very effective for nearest neighbor queries, and are also generally very good.**
- **R-trees may be less effective for some data sets, and more effective for others.**
- **In Oracle9i Spatial, some features are only supported via R-trees (incremental nearest neighbor, whole Earth indexing).**

ORACLE

7-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Quadtrees or R-trees? Choosing an Index

Some important points to understand when choosing an index type include:

- Fixed quadtree indexes are very good in nearly all cases regardless of the data being indexed and the types of queries run, but require tuning expertise to determine the right tile size for optimal query performance. This lesson provides the information required to help users determine what the best level is for a given spatial layer.
- R-trees are very good for most query types, and are the best index to use for nearest neighbor queries.
- There are some data sets that R-trees may not be the best index for, for example a data set where the MBR around all of the geometries is almost all empty space.
- Some features are only supported using R-tree indexes, including incremental nearest neighbor and whole Earth indexing.

To simplify index creation and give access to all Oracle Spatial functionality, Oracle is working to make R-tree indexes as good as or better than quadtree indexes in as many cases as possible.

Tuning Assistants

The SDO_TUNE Package

- **SDO_TUNE.EXTENT_OF**
 - Determines the minimum bounding rectangle of the data in a layer
- **SDO_TUNE.ESTIMATE_TILING_LEVEL (Quadrees)**
 - Gives a starting point to help determine an appropriate tiling level (SDO_LEVEL) for creating fixed-size index tiles

ORACLE

7-5

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_TUNE Package

Two functions typically used to help determine the starting SDO_LEVEL when creating a fixed quadtree spatial index are:

- **SDO_TUNE.EXTENT_OF**: This function returns a minimum bounding rectangle around all of the data in a spatial layer.
- **SDO_TUNE.ESTIMATE_TILING_LEVEL**: This function is used to give an appropriate value for a starting point for SDO_LEVEL when creating a fixed size tile quadtree index.

EXTENT_OF

```
<geometry> := SDO_TUNE.EXTENT_OF  
  ( <table-name>, <column-name> )
```

- **<table-name> and <column-name> identify the spatial layer.**
- **<geometry> returns a rectangle that defines the extent (or minimum bounding rectangle) of the layer.**

ORACLE

7-6

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_TUNE.EXTENT_OF

SDO_TUNE.EXTENT_OF is called with two literal string parameters. The <table-name> parameter is a string literal which specifies the name of the table that the spatial layer is in, and the <column-name> parameter is a string literal that identifies the column of type MDSYS.SDO_GEOMETRY which will be examined by the function.

The function returns a <geometry>, which is of type MDSYS.SDO_GEOMETRY. The geometry returned is an optimized rectangle that contains the minimum and maximum bounds for the layer. The geometry returned is always has two dimensions regardless of the number of dimensions in the input layer.

A subsequent slide discusses how the SDO_TUNE.ESTIMATE_TILING_LEVEL function can use SDO_TUNE.EXTENT_OF to recommend a quadtree spatial indexing level for the layer.

EXTENT_OF Example

```
SELECT
    sdo_tune.extent_of('PROJ_STATES', 'GEOM')
FROM dual;

SDO_GEOMETRY(2003, NULL, NULL,
    SDO_ELEM_INFO_ARRAY(1, 1003, 3),
    SDO_ORDINATE_ARRAY(-10426096, -78093.788,
        3407718.72, 6746612.21))
```

ORACLE

7-7

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_TUNE.EXTENT_OF (continued)

The query in this example will return the minimum bounding rectangle for the spatial layer in column GEOM which is in table PROJ_STATES. As shown, the query returns a geometry object which is the minimum bounding rectangle that encompasses all the geometries in the GEOM column. The returned geometry object is specified in optimized rectangle format.

Note that the table specified in the FROM clause in the above statement is table DUAL. Table DUAL is a special Oracle-specific table that has two interesting properties:

- It is guaranteed to exist in every instance of Oracle
- It is guaranteed to only have one row loaded in the table.

In the above case a select FROM table DUAL is used to ensure that the query is only executed once. If another table was used, for instance the PROJ_STATES table, then the query would have executed once for each row in the PROJ_STATES table (56 times), returning the exact same minimum bounding rectangle each time.

Flexible Fixed Indexing (Quadtree)

- If query window comes from a table and is indexed at same level as layer, index is reused
- If query window comes from a table and is not indexed at same level as layer, query window is indexed in memory
- If the query window requires indexing in memory at query time, a maximum of 4K tiles will be generated for that window. If more than 4K tiles will be generated, the SDO_LEVEL will be reduced

ORACLE

7-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Flexible Fixed Indexing for Quadtrees

Oracle Spatial has a flexible indexing model. In the first release of the object-relational model of Oracle Spatial (Oracle 8.1.5) there was a fixed limitation on the number of tiles allowed for a window geometry. Any time the tessellation of a window geometry generated more than 30,000 tiles, the indexing of the geometry would fail with an error message. Now Oracle Spatial has a very flexible model for indexing geometries.

If the query window comes from a table which is indexed at the same level as the table layer, the existing index is reused. This results in the best performance because the query window does not need to be indexed.

If the query window comes from a table but is not indexed at the same level as the layer from the table, then the query window is indexed in memory.

If the query window is not indexed (it could be a variable or from a table that isn't indexed) Oracle Spatial will index it in memory.

In either of the two previous cases where the query window is indexed in memory at query time, Oracle Spatial will attempt to index the window geometry at the same level as the layer. However, there is a fixed limit on the number of tiles that will be generated for the query window. This limit is 4K tiles. If more than 4K tiles would be generated, then the SDO_LEVEL is reduced.

ESTIMATE_TILING_LEVEL (Quadtree Only)

```
level := SDO_TUNE.ESTIMATE_TILING_LEVEL  
( <table-name>, <column-name>,  
  <number-of-tiles>,  
  <type-of-estimate> )
```

- **<table-name> and <column-name> identify the spatial layer**
- **<number-of-tiles> is the number of tiles desired**
- **<type-of-estimate> defines the type of estimation to perform**

ORACLE

7-9

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_TUNE.ESTIMATE_TILING_LEVEL

Estimate tiling level is a utility that you will probably never call directly. First let's discuss the theory behind ESTIMATE_TILING_LEVEL. Prior to the flexible fixed index, it was very important to pick a tiling level that would ensure any query window would not generate too many tiles. This is no longer necessary (due to the flexible fixed index), but this utility is a very effective way to get a good (but not necessarily optimal) fixed tiling level. The Oracle Enterprise Manager (OEM) Spatial Index Advisor calls ESTIMATE_TILING_LEVEL to get a good guess at what SDO_LEVEL to use. OEM Spatial Index Advisor then does a much more scientific analysis (that is, sampling) to determine an optimal fixed tiling level. ESTIMATE_TILING_LEVEL is discussed here to give you a better understanding of how OEM Spatial Index Advisor uses this utility.

The ESTIMATE_TILING_LEVEL function returns a value that represents the tiling level to start with based on one of three types of estimations that is chosen by the user.

ESTIMATE_TILING_LEVEL provides an estimation or starting point for indexing. This function recommends a tile level based on an educated guess, and the user may attempt to increase performance by choosing a numbers near the recommended estimate. For example, if the function returns an SDO_LEVEL of 7 and the primary filter queries are not selective enough, an SDO_LEVEL=8 tile may produce better results. SDO_LEVEL 8 tiles are one-quarter the size of SDO_LEVEL 7 tiles.

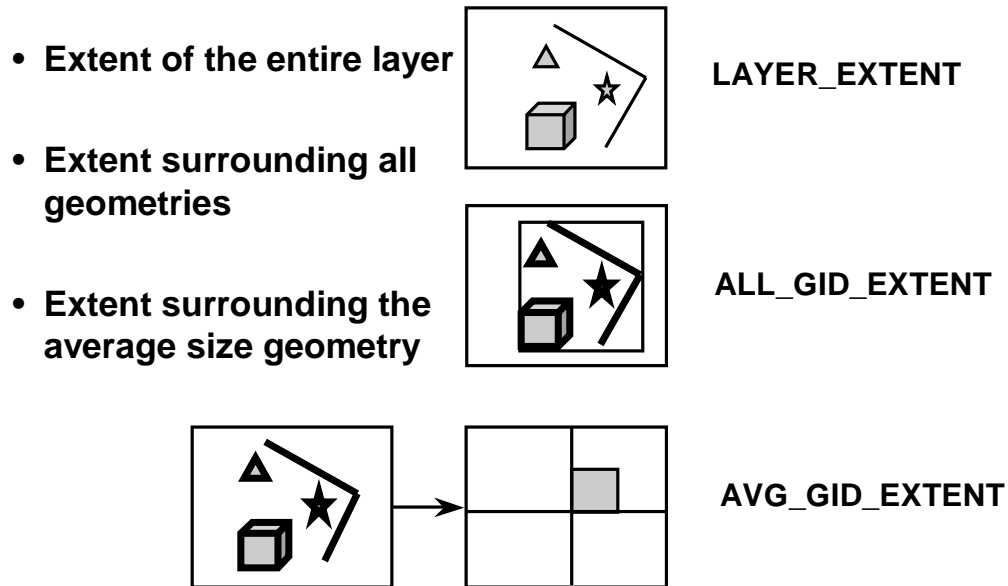
SDO_TUNE.ESTIMATE_TILING_LEVEL (continued)

Input parameters to the ESTIMATE_TILING_LEVEL function include:

- <table-name> A string literal that contains the name of the table the spatial layer is in
- <column-name> A string literal that contains the name of the geometry column
- <number-of-tiles> An integer value specifying the number of tiles desired for the estimate
- <type-of-estimate> A string literal which is one of three predefined strings that defines the type of estimate to perform. These three types are defined in the next slide.

In all cases, Oracle Spatial requires that the USER_SDO_GEOM_METADATA view be populated before running the ESTIMATE_TILING_LEVEL function.

Type of Estimation



ORACLE

7-11

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_TUNE.ESTIMATE_TILING_LEVEL: Types of Estimation

There are three types of estimations used for ESTIMATE_TILING_LEVEL. The type of estimate will determine the rectangle the analysis will be carried out on.

- **LAYER_EXTENT**: Performs analysis on the rectangle that represents the extent defined in the USER_SDO_GEOM_METADATA view. The estimation calculates the level required to generate no more than <number-of-tiles> for the analysis rectangle.
- **ALL_GID_EXTENT** (recommended): Performs analysis on the rectangle returned from the SDO_TUNE.EXTENT_OF function for the spatial layer. The estimation calculates the level required to generate no more than <number-of-tiles> for the analysis rectangle.
- **AVG_GID_EXTENT**: This estimation looks at every geometry and determines the average sized minimum bounding rectangle that would cover a single geometry. This MBR is then transposed to a corner of the coordinate system as stored in the USER_SDO_GEOM_METADATA view, and the SDO_LEVEL is calculated based on the average MBR and the <number-of-tiles> specified by the user to cover that MBR.

ESTIMATE_TILING_LEVEL Example

```
select sdo_tune.estimate_tiling_level (
  'PROJ_STATES', 'GEOM', 10000, 'LAYER_EXTENT')
from dual;

select sdo_tune.estimate_tiling_level (
  'PROJ_STATES', 'GEOM', 10000, 'ALL_GID_EXTENT')
from dual;

select sdo_tune.estimate_tiling_level (
  'PROJ_STATES', 'GEOM', 4, 'AVG_GID_EXTENT')
from dual;
```

ORACLE

7-12

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_TUNE.ESTIMATE_TILING_LEVEL Examples

Three examples of the use of SDO_TUNE.ESTIMATE_TILING_LEVEL are shown.

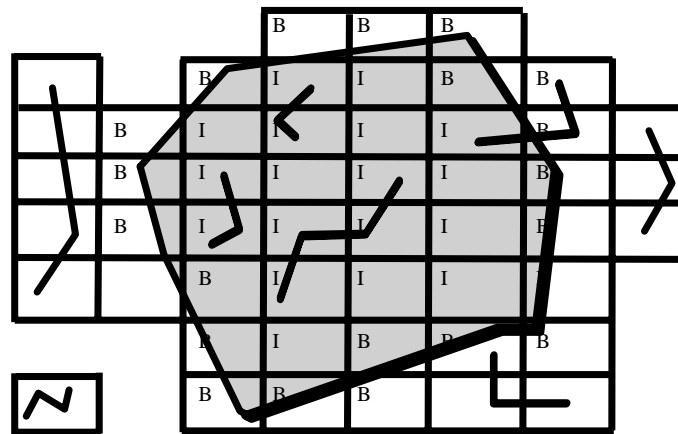
The first determines the tiling level for the GEOM column in the PROJ_STATES layer, calculating the SDO_LEVEL required to generate no more than 10000 tiles using 'LAYER_EXTENT' (the extent of the entire coordinate system).

The second query uses 'ALL_GID_EXTENT', so using the GEOM column in the PROJ_STATES table it creates the minimum bounding rectangle that encompasses all of the geometries in the layer, then determines what tiling level would be required to generate no more than 10000 tiles within the minimum bounding rectangle of the data, in the context of the larger coordinate system extent.

The last query uses 'AVG_GID_EXTENT'. The query will determine the SDO_LEVEL required to generate no more than 4 tiles for the average sized MBR around each geometry in the GEOM column of the PROJ_STATES layer.

Note: Oracle recommends using ALL_GID_EXTENT with <number-of-tiles> = 10000.

Primary and Secondary Filter Concept Quadtree Example



ORACLE

7-13

Copyright © Oracle Corporation, 2001. All rights reserved.

Primary and Secondary Filter Concept

Let's say there are 20 million roads (linestrings) in a layer and a query is run to determine which roads interact with a zip code polygon. If there were no spatial index, computational geometry to compare each line segment of each road to the perimeter and interior of the polygon to see if there is any interaction would be required. This would require a full scan of all the roads.

An important task of a spatial index is to reduce the number geometries that need to be compared. Comparing the line segments and interiors of geometries is a very expensive process with respect to time. The primary filter only compares tiles, not the true geometries. In this example, a quadtree spatial index was used. For each road, quadtree tiles were generated to approximate the road. Quadtree tiles were also generated to approximate the polygon.

Since only approximations are compared in the primary filter, it will return a superset of the true result. It returns the entire answer, and possibly a few more geometries. The primary filter will return all the roads that have a tile that is the same as one of the tiles that approximates the polygon. In this case, five yellow roads returned in the primary filter (four that truly interact, and one that does not).

Primary and Secondary Filter Concept (continued)

Oracle Spatial gives the option of just performing a primary filter. A primary filter will give all the geometries that satisfy the query, and possibly additional geometries. Some applications don't need an exact answer (for example, ZOOM IN/OUT on a mapping viewport). The ZOOM IN/OUT can perform a primary filter, and use its clipping routines to discard the additional geometries that returned from the primary filter.

If the exact answer is desired a secondary filter query can be run, which only compares the coordinates of geometries that are returned in the primary filter, not all 20 million roads. This will result in the exact answer.

In addition, Oracle Spatial applies an interior tile optimization to further improve secondary filter performance. Tiles for the polygon are tagged as interior (I) or boundary (B). If a road tile equals the polygon's I tile, that geometry can get accepted as having passed the secondary filter during the primary filter process (no need to compare the line segments of the geometries). This optimization is done for all secondary filter masks (for example, TOUCH relationships will only evaluate roads that touch B tiles).

A similar interior tile optimization is also done with R-trees. Quadtree indexes require tuning for the interior tile optimization. This optimization is transparent with R-tree indexes, and is not part of the tuning process.

Oracle Enterprise Manager (OEM) Spatial Index Advisor

- **Oracle Spatial Index Advisor: Tuning utility ships with OEM (Oracle Enterprise Manager)**
- **An Oracle EE typical install will install the Spatial Index Advisor**
- **Before using, grant SELECT ANY DICTIONARY to any user that will create an index using Spatial Index Advisor (was SELECT_CATALOG_ROLE in 8.1.7)**
- **Also need to set up connection information using OEM console before using Spatial Index Advisor**
- **To start Spatial Index Advisor, enter: oemapp sdoadvisor**

ORACLE

7-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Advisor

Spatial Index Advisor is a quadtree index tuning utility that ships with Oracle Enterprise Manager. A typical Enterprise Edition install will automatically install Spatial Index Advisor.

Before using Spatial Index Advisor to create an index, the SELECT ANY DICTIONARY privilege must be granted to the user who will create the index. An example of this (run as SYSTEM) is:

```
GRANT SELECT ANY DICTIONARY TO SCOTT;
```

Also, starting in Oracle9i users will have to set up OEM to work with the database using OEM console. To do this:

1. Enter oemapp console, then click “Launch standalone,” then click “OK.
2. Click the menu item Navigator, then click Add Database to Tree.
3. Click the button to “Add selected databases from your tnsnames.ora file,” leave selected the service name associated with the database to which you wish connect, then click OK.

Spatial Index Advisor (continued)

4. Next click the “Configuration” menu item, select the service name that was just added by clicking on it, enter the username and password (for the course the username is SCOTT and the password is PW4SCOTT), then click OK.

At this point, assuming the tnsnames.ora configuration is correct (this can be tested using SQL/Plus), the Spatial Index Advisor can be started. This can be done from the console by clicking the Database Applications icon on the left menu bar, then clicking the Spatial Index Advisor icon, or from the command line you can type `oemapp sdoadvisor`.

Oracle Enterprise Manager (OEM) Spatial Index Advisor

DEMO

ORACLE

7-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Advisor Demo

In this demonstration, Spatial Index Advisor (and the quadtree indexing wizard) is used to help create and tune quadtree spatial indexes. This tool can also be used to create R-tree indexes (R-tree indexes do not require tuning).

The demonstration starts with the PROJ_STATES layer which was created, but not indexed in a previous Lab. The PROJ_STATES layer will be used as a reference layer to help find dense areas in other layers. Each individual layer should be tuned individually using Spatial Index Advisor.

After starting Spatial Index Advisor the Add Layer dialog comes up. Click on the database name, then click on SCOTT, then click on the PROJ_STATES table, then click OK. The message on the banner will tell you that a nonindexed layer has been added, and there are only certain functions within the Spatial Index Advisor that are enabled.

Spatial Index Advisor Demo (continued)

Click Index > Create Index on the top menu bar. This will bring up Schema Manager. In Schema Manager, click Show SQL on the bottom of the page. In the Name field, enter an index name. You can optionally specify Schema and Tablespace information for the index. Click on the table column of type SDO_GEOMETRY (in the PROJ_STATES table, it is the GEOM column). Note a tab with Spatial appears along the top. Before clicking on the Spatial tab, first click on Storage to set storage parameters. This isn't necessarily required for small indexes like the ones created for the course, but they are very useful for larger indexes. For the course, they can be set to an initial size of 2 MB, a next size of 1 MB, and an increase size by 0%. Next, click the Spatial tab, then click on Quadtree. If the spatial data is point only data then click on the appropriate check box. Next, click Estimate... to open the Tiling Wizard.

The Welcome page gives an overview of the Tiling Wizard. Based on the data, the Tiling Wizard uses a combination of Sampling, Area of Interest, and Tile Size Analysis to help determine a good initial tiling level for a quadtree index. Click Next, and if the data is exclusively point data click the appropriate option button. The next page is the Sampling Analysis page. On this page you choose the percentage of data to sample for analysis. The Tiling Wizard uses sampling analysis to build indexes on the sampled data to try to determine an optimal tiling level. The default sample size is 1,000 rows. The sample percentage can be adjusted manually. If the data is not geodetic, the next page is Area of Interest Analysis. With Area of Interest Analysis, a typical sized query window can be chosen and a tiling level can be calculated based on the size of a typical query window. This analysis is only useful for area consistent coordinate systems (such as projected coordinate systems), hence the analysis is not allowed if the data is geodetic (has a geodetic SRID). The next window is the Tile Size Analysis. This runs the SDO_TUNE.ESTIMATE_TILING_LEVEL function discussed in a previous slide. The default (which is recommended) is 10,000 tiles to cover the MBR of the geometries in the layer. Sampling is not used for this function; the function reads all of the geometries in the layer. The final page is the Finish page, which recaps the choices made and what types of estimates will be done. Click Finish, and the Tiling Wizard will do the analysis.

The first analysis completed (if selected) is the Area of Interest Analysis. If there were no other considerations (such as size of the index table) then this estimate based on the size of the query window would give the final answer for the tiling level, and no additional analysis would be required. The next analysis done is the Tile Size Analysis. This gives the tiling level determined (in the example above) by the level required to get 10000 tiles to cover the minimum bounding rectangle around all of the geometries in the layer. The next analysis is the sampling analysis, which is done on the data sample. Sampling Analysis will continually index your sample data with larger SDO_LEVELs (smaller tiles). As the SDO_LEVEL increases, the Advisor compares how many tiles were generated at the previous SDO_LEVEL to the number of tiles generated at the current SDO_LEVEL. The advisor is determining how small a tile it can use without making your spatial index table grow too large. Layers consisting of only points do not require sampling analysis because each point will always generate one tile. The starting SDO_LEVEL for the sampling analysis is the result of the tile size analysis. The result of the Sampling Analysis is the initial tiling level that should be used to create the index. Click close.

The Create Index window now has the Initial tiling level filled in. Click Create to create the index. Further analysis to complete follows.

Oracle Enterprise Manager (OEM) Spatial Index Advisor

- Remember to **ANALYZE** your quadtree spatial index tables. This is *required* for optimal performance. Here is some SQL that generates SQL to compute statistics.

```
SQL> select 'ANALYZE TABLE ' || SDO_INDEX_TABLE ||  
          ' COMPUTE STATISTICS; '  
from user_sdo_index_info  
where sdo_index_type = 'QTREE';
```

- For very large quadtree spatial index tables, you can estimate 1 percent (if you don't have the time to compute statistics).

```
SQL> select 'ANALYZE TABLE ' || SDO_INDEX_TABLE ||  
          ' ESTIMATE STATISTICS SAMPLE 1 PERCENT; '  
from user_sdo_index_info  
where sdo_index_type = 'QTREE';
```

ORACLE

7-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Analyze Indexes

After creating quadtree spatial indexes it is important to analyze the spatial index table using Oracle's ANALYZE TABLE command. This is required for optimal performance. Completing this operation adds statistics and other information to the Oracle dictionary tables, enabling the Oracle optimizer to choose an optimal execution plan when the spatial index table is accessed.

The statements shown will generate ANALYZE TABLE commands. The ANALYZE TABLE commands must then be cut and pasted back into SQL*Plus to be executed by Oracle. The first SQL statement shown generates the following SQL statement for each quadtree spatial index table:

```
ANALYZE TABLE INDEX_TABLE_NAME COMPUTE STATISTICS;
```

Computing statistics can take a long time if the spatial index table is very big. Estimating statistics is a way of minimizing the amount of time required to gather statistics. With versions of Oracle from 8.1.6 and beyond it has been demonstrated that sampling as little as 1 percent of the spatial index table is effective to ensure the optimizer uses the index table properly. The second SQL statement shown generates the following SQL statement for each quadtree spatial index table:

```
ANALYZE TABLE INDEX_TABLE_NAME ESTIMATE STATISTICS SAMPLE 1  
PERCENT;
```

Again, note that the SQL statement shown in the slide generates SQL that must then be run in Oracle to take effect.

Oracle Enterprise Manager (OEM) Spatial Index Advisor

Iterative Process (for quadtree indexes)

- **Create quadtree index**
- **View data at dense location**
- **Zoom to reasonable size**
 - If too large an area, may need to choose a higher number **SDO_LEVEL** (smaller tiles)
 - If too small an area, may need to choose a lower number **SDO_LEVEL** (larger tiles)

ORACLE

7-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Iterative Quadtree Index Tuning Procedure

Tuning quadtree spatial indexes is an iterative process. The Oracle Spatial Index Advisor is used to help determine a good starting point for the **SDO_LEVEL** to choose when building an index, but there is no guarantee the value chosen will be the best for a specific application. Oracle recommends that when using Oracle Spatial with quadtree indexes, use the Tiling Wizard in Schema Manager as previously presented. Additional analysis to verify the index is optimal should also be done. The methodology used for determining if the quadtree index is optimal includes the following steps:

First, view the data at a dense location. Dense locations are used because these are the areas where highest performance is required. If the performance is good in dense areas, then the performance will be good throughout the entire data set. To view the data at a dense location, use a base layer to find dense locations in another layer. For example, detailed street data is very dense in New York City. Use a STATES layer (that is, your base layer) to navigate to New York City, then change your base layer to the streets layer. Now click View > Zoom to > Reasonable Size and click in the center of the densest area. The Spatial Index Advisor will then zoom to a viewport where the size of the query window will include approximately 200 interior tiles. The tiles can be seen by clicking Index > Show Fixed-Size Tiles.

Iterative Quadtree Index Tuning Procedure (continued)

If the Zoom to Reasonable Size operation shows too large an area compared to a typical query window, then increase the SDO_LEVEL and recreate the index to get smaller tiles. When the tiles are smaller, good interior tile optimizations can be done on a smaller area. If Zoom to Reasonable Size shows too small an area, then reduce the SDO_LEVEL and recreate the index to get larger tiles. In this way, a larger query window will be able to make use of interior tile optimizations as well.

Note: To increase or decrease the tiling level, you do not have to use the tiling wizard in schema manager. You can make an educated guess. Increasing the SDO_LEVEL by 1 will reduce the tile size by 1/4 (smaller tiles). Decreasing the SDO_LEVEL by 1 will increase the tile size by four times (larger tiles).

Oracle Enterprise Manager (OEM) Spatial Index Advisor

Additional checks at dense areas

- Look at tiles (Show Fixed-size Tiles) on left menu bar
- Use Layer > Query > and select a shape or a geometry to query that represents a typical query window
- Select Primary Filter Only to show the results of the index query only, then click Apply
- Select Spatial Interaction with ANYINTERACT, then click Apply
- Note the effectiveness of the primary filter. Did it return similar data to the Spatial Interaction with ANYINTERACT?

ORACLE

7-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Iterative Quadtree Index Tuning Procedure

Additional checks can be done to determine how efficient the index is for a given query.

In a dense area, show the index tiles stored as a result of the spatial indexing operation.

Next, click on Layer > Query and choose a shape that represents a typical query window. Draw the shape, and the next dialog box is for the query window. Select Primary Filter Only, and click Apply. The results of this query will be all of the geometries based on comparing approximations of geometries, that is, comparing all of the tiles of the query window just entered with all of the tiles in the PROJ_STATES layer. Observe the number of geometries returned. Now try the same query with the Spatial Interaction set to ANYINTERACT, then click Apply. The results of this query only returns geometries that interact with the query window.

Iterative Quadtree Index Tuning Procedure (continued)

This query produces a result set in three steps:

- The spatial index is used to return approximate results based on the geometry approximations (tiles).
- Some geometries are known to have an interaction with the query window based on interior tile optimizations (see earlier slide). These geometries are passed through to the final result set.
- Some geometries are passed to the secondary filter for full geometric computational comparisons.

Note the performance of the ANYINTERACT query. Does it compare favorably to the time required to do the Primary Filter query? Note the number of geometries returned. Is it close to the same number of geometries returned for the primary filter query?

If performance is good and the primary filter is doing an effective job in terms of eliminating geometries based on the index, then the quadtree index is tuned well for a typical query window.

Summary

In this lesson, you should have learned how to:

- **Determine the appropriate tiling level when building a quadtree index on a spatial layer**
- **Use the OEM tool Spatial Index Advisor to create and tune spatial indexes**

ORACLE

8

Spatial Queries

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Describe spatial window and join queries**
- **Explain the Oracle Spatial query model**
- **Describe and compare spatial operators and functions**
- **Use spatial operators and functions to perform spatial queries**
- **Understand the topological relationships used by spatial operators and functions**

ORACLE

8-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

In this lesson you will learn:

- The difference between spatial window queries and spatial join queries
- The query model used by Oracle Spatial (primary and secondary filters)
- The differences between spatial operators and spatial functions
- How to write queries that use spatial operators and spatial functions
- The different topological relationships supported by spatial operators and functions

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

8-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Data Sets Loaded So Far

So far you have loaded eight layers. Four of the layers are defined using a geodetic coordinate system:

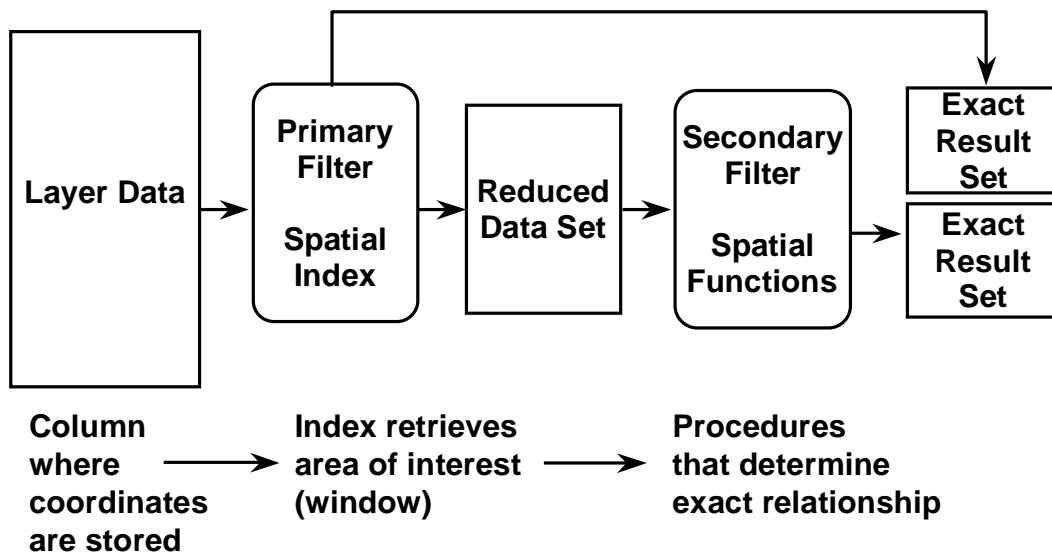
- **GEOD_STATES**
- **GEOD_COUNTIES**
- **GEOD_INTERSTATES**
- **GEOD_CITIES**

and four of the layers are in a projected coordinate system:

- **PROJ_STATES**
- **PROJ_COUNTIES**
- **PROJ_INTERSTATES**
- **PROJ_CITIES**

The prefixes **GEOD_** and **PROJ_** will help you recognize whether a query against a geodetic layer or a projected layer is being demonstrated.

Optimized Query Model



ORACLE

8-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Optimized Query Model

This slide describes the Oracle Spatial optimized query model. Start out with a Spatial layer and ask a spatial question (for example, show 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.

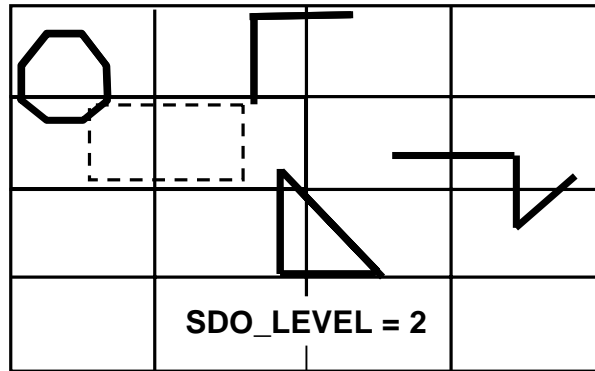
When you create a spatial index, the spatial index stores geometry approximations for each geometry. The primary filter portion of the query compares the geometry approximations instead of the true geometries, and generates a reduced data set. Comparing geometry approximations is much faster than comparing the true geometries. There are some optimizations Oracle does in the primary filter, where it can accept geometries as truly interacting, even though only geometry approximations are compared. This is why you see some results going to the exact result set, and some going to the reduced data set from the primary filter stage.

Oracle Spatial gives you the option of just performing a primary filter. A primary filter will give you all the geometries that satisfy a query, and possibly some additional geometries. Some applications don't need an exact answer, for example, ZOOM IN/OUT on a mapping viewport. The ZOOM IN/OUT can perform a primary filter, and use its clipping routines to discard the additional geometries that returned from the primary filter.

Optimized Query Model (continued)

If the exact answer is desired, for example, if you drew a circle on the viewport and asked for all the parcels that interacted with the circle, you probably don't want a fuzzy answer (that is, a few more parcels than the true answer). For the exact answer, you can run a secondary filter, which compares only the coordinates of the geometries that were returned in the primary filter that were not automatically accepted as exact, not the entire layer. The geometries returned in this exact comparison added to the geometries accepted as exact in the primary filter stage will result in the exact answer.

Primary Filter Example (Quadtree Index)



- **Compares fixed sized tiles that approximate the area of interest with fixed sized tiles that approximate each geometry**
- **Result is not exact because comparing approximations**

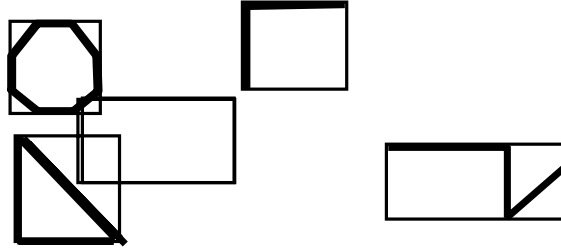
Primary Filter Example Using a Quadtree Index

In this example, the index is being used to determine which geometries have any interaction with the rectangular geometry whose boundary is represented by the dotted line.

The quadtree spatial index is used to store approximations of each of the geometries shown. Every tile that covers part of each geometry is stored in the spatial index along with a pointer to that geometry. The query window (the rectangle whose boundary is a dotted line) is also spatially indexed. There are two tiles associated with the dotted rectangle. The spatial index works by comparing approximations of geometries. In this quadtree index case, the approximations are tiles. All geometries that have an index entry that includes either of the two tiles that approximate the rectangle are returned by the spatial index query. In this example, three geometries are returned by the spatial index.

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 (or primary filter) will always return the correct final results of a spatial query, and perhaps some extra geometries as well.

Primary Filter Example (R-tree Index)



- **Compares MBR that approximates the area of interest with the MBR that approximates each geometry**
- **Result is not exact because comparing approximations**

Primary Filter Example Using an R-tree Index

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

The R-tree spatial index is used to store approximations of each of the geometries shown. Every minimum bounding rectangle that covers each geometry is stored in the spatial index along with a pointer to that geometry. The query window (the rectangle) is also spatially indexed. The spatial index works by comparing approximations of geometries. In this R-tree index case, the approximations are minimum bounding rectangles. All geometries that have an index entry whose minimum bounding rectangle interacts with the minimum bounding rectangle that approximates the query window are returned by the spatial index query. In this example, two geometries are returned by the spatial index.

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 (or primary filter) will always return the correct final results of a spatial query, and perhaps some extra geometries as well.

Spatial Operators versus Functions

- **Spatial Operators:**
 - Take advantage of spatial indexes
 - Require that spatial index exists on the first geometry specified in the operator
 - Only in the WHERE clause
- **Spatial Functions:**
 - Do not take advantage of spatial indexes
 - Could be used on small tables that are not spatially indexed
 - Can be used in the SELECT list or WHERE clause

ORACLE

8-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Operators versus Spatial Functions

Spatial operators take advantage of spatial indexes, and spatial functions do not. Spatial operators require that an index is built on the first geometry (the search column) specified in the operator. Spatial operators can only be used in the WHERE clause of a SQL statement.

Spatial functions do not take advantage of spatial indexes. They can be used on small tables that are not spatially indexed. Spatial functions can be used both in the SELECT list and the WHERE clause of a SQL statement.

Spatial Operators versus Functions

Operators

- **SDO_FILTER**
 - Performs a primary filter only
- **SDO_RELATE**
 - Performs a primary and secondary filter
- **SDO_WITHIN_DISTANCE**
 - Generates a buffer around a geometry and performs a primary and optionally a secondary filter
- **SDO_NN**
 - Returns nearest neighbors

Functions

- **SDO_GEOM.RELATE**
 - To determine the relationship between two geometries
 - To perform a spatial query without using a spatial index (for example, on a small table)
- **SDO_GEOM.WITHIN_DISTANCE**
 - Generates a buffer around a geometry and performs a secondary filter. This is almost never used (use `sdo_within_distance` instead).

Spatial Operators versus Spatial Functions

SDO_FILTER

Uses the spatial index to compare geometry approximations in a spatial query to identify pairs of geometries that potentially interact. Geometries interact spatially if they are not disjoint. Because **SDO_FILTER** only compares geometry approximations, it returns all the candidates that actually interact, and possibly a few more.

SDO_RELATE

Uses the spatial index to identify either geometries that have a particular spatial interaction with a given area of interest geometry, or pairs of geometries that have a particular spatial interaction. This operator performs both primary and secondary filter operations.

SDO_WITHIN_DISTANCE

Uses the spatial index to identify the set of geometries that are within some specified distance of a given geometry, such as an area of interest or a point of interest.

SDO_NN

Uses the spatial index to identify the nearest neighbors of a geometry.

Spatial Operators Versus Spatial Functions (continued)

SDO_GEOM.RELATE

Does not use a spatial index. Examines two geometry objects to determine their spatial relationship. This function is almost never used in the WHERE clause (SDO_RELATE is used instead). SDO_GEOM.RELATE can be useful in the SELECT clause to report back the relationship between two geometries.

SDO_GEOM.WITHIN_DISTANCE

Does not use a spatial index. Determines if two spatial objects are within some specified distance from each other. This function is almost never used. Use SDO_WITHIN_DISTANCE instead to take advantage of a spatial index.

The SDO_FILTER Operator

```
SDO_FILTER  
( <geometry-1>,  
  <geometry-2>,  
  'QUERYTYPE=<querytype>  
    [other optional parameters]'  
) = 'TRUE'
```

- Performs an approximate query (primary filter only)
- Returns TRUE or FALSE

ORACLE

8-11

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_FILTER Operator

Uses the spatial index to compare geometry approximations in a spatial query to identify pairs of geometries that potentially interact. Geometries interact spatially if they are not disjoint. Since SDO_FILTER only compares 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, arg3) = 'TRUE'.

Think of this spatial operator's syntax as a template. If you understand this spatial operator's syntax, you will easily understand the other spatial operators, which all have similar syntax.

The first geometry in the parameter list is the column you are searching. The second geometry in the parameter list is the window (or area of interest). The third parameter is a quoted string of parameters specific to the operator.

Required Arguments

- **GEOMETRY-1**
 - Must be a column in a table
 - Must be of type **SDO_GEOMETRY**
 - Must be indexed
- **GEOMETRY-2**
 - Variable or column in a table
 - Must be of type **SDO_GEOMETRY**
 - If **querytype=JOIN** (very rare), must be a column in a table and for best performance indexed with the same index type (quadtree/R-tree) and at the same **SDO_LEVEL** (if quadtree)
- **QUERYTYPE**
 - Valid values are **WINDOW** or **JOIN** (very rare)

ORACLE

8-12

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_FILTER Syntax

GEOMETRY-1

Specifies a geometry column in a table. The column must be spatially indexed and of the data type **MDSYS.SDO_GEOMETRY**. This is the spatial column being searched.

GEOMETRY-2

Specifies either a geometry from a table or a transient instance of a geometry (specified using a bind variable or **SDO_GEOMETRY** constructor). Its data type is **MDSYS.SDO_GEOMETRY**.

QUERYTYPE

Specifies valid query types: **WINDOW** or **JOIN**. This is a required parameter. **WINDOW** is recommended in almost all cases.

WINDOW implies that a comparison is performed between the index entries associated with *geometry2* (the query window or area-of-interest) and each index entry for the layer associated with *geometry1*.

SDO_FILTER Syntax (continued)

The extreme example of a WINDOW constrains *geometry2* to a single geometry, but window queries do not limit *geometry2* to a single geometry.

JOIN is rarely used, and not recommended. JOIN compares all the geometry index records of a column to all the geometry index records of another column (you can think of this as a spatial cross product). JOIN requires spatial indexes exist on both parameters, *geometry1* and *geometry2*.

Any JOIN query can be rewritten as a WINDOW query which will be demonstrated in an upcoming slide.

Optional Arguments

- **IDXTAB1**
 - Index table to associate with first geometry in operator
 - By default, the primary index table is used
- **IDXTAB2**
 - Index table to associate with the second geometry in operator
 - By default, the primary index table is used
 - Only supported if QUERYTYPE=JOIN

ORACLE

8-14

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_FILTER Syntax

IDXTAB1

Specifies the name of the spatial index table to use for *geometry1*, if there are multiple spatial index tables for *geometry1*. By default, the primary spatial index table is used.

IDXTAB2

Specifies the name of the spatial index table to use for *geometry2*, if there are multiple spatial index tables for *geometry2*. By default, the primary spatial index table is used. IDXTAB2 is valid only if *querytype* is JOIN (which is not recommended).

Optional Arguments

LAYER_GTYPE (deprecated in Oracle9i)

- Before Oracle9i, set to POINT if querying POINT only columns (for optimal query performance)
- Otherwise, NOTPOINT (default)
- In Oracle9i and later, set the LAYER_GTYPE parameter in the CREATE INDEX command instead

ORACLE

8-15

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_FILTER Syntax

The LAYER_GTYPE keyword has been deprecated in this operator, and it is ignored if specified.

Note: Specify the LAYER_GTYPE when creating a spatial index, as one of the CREATE INDEX parameters.

SDO_FILTER Example

- Find all cities in a selected rectangular area
- Result is approximate

```
select c.city, c.pop90
from proj_cities c
where sdo_filter (
    c.location,
    mdsys.sdo_geometry (2003, 32775, null,
        mdsys.sdo_elem_info_array (1,1003,3),
        mdsys.sdo_ordinate_array (1720300,1805461,
                                   1831559, 2207250)),
    'querytype=WINDOW') = 'TRUE';
```

Hint 1: All Spatial operators return TRUE or FALSE. When writing spatial queries always test with = 'TRUE', never <> 'FALSE' or = 'true'.

ORACLE

8-16

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_FILTER Example

This query finds all the cities in a selected rectangular area, giving an approximate result. The result is approximate because only a primary filter operation is being performed.

A brief description of the SDO_FILTER parameters:

geometry1 -- the search column -- is C.LOCATION from the PROJ_CITIES table

geometry2 -- the window -- is a geometry constructor

querytype=WINDOW

Hint 1

All Spatial operators return TRUE or FALSE. When writing spatial queries specify = 'TRUE'; never <> 'FALSE' or = 'true'

SDO_CS.VIEWPORT_TRANSFORM

- **Previous query fine for projected layers**
- **There are geodetic layer restrictions**
 - **No optimized rectangles**
 - **No rectangle with surface area equal to or greater than half the surface area of the earth**
- **SDO_CS.VIEWPORT_TRANSFORM is a convenience function used with any optimized rectangle which returns a valid polygon for any given coordinate system**

ORACLE

8-17

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.VIEWPORT_TRANSFORM

As discussed in the “Coordinate Systems Overview” lesson, geodetic layers in Oracle Spatial have some restrictions. One restriction is that optimized rectangles are not allowed, and another is polygons must always be less than half the surface area of the earth (refer to the “Coordinate Systems Overview” lesson for a complete discussion on the restrictions and why they are imposed).

This function is very useful for visualizer applications. Visualizer applications often display geodetic data as a regular Cartesian coordinate system (earth displayed as a rectangle on the screen). Typical visualizer applications often perform rectangular queries on geodetic layers, for example when zooming in and out.

SDO_CS.VIEWPORT_TRANSFORM Function

```
SDO_GEOMETRY :=  
SDO_CS.VIEWPORT_TRANSFORM (geom, to_srid);
```

- **geom = geometry of type SDO_GEOMETRY**
 - can be a variable or table column
 - must be an optimized rectangle with SRID set to 0
- **to_srid = Spatial reference system ID to transform to**
- **returns =**
 - If <to_srid> is a geodetic SRID, returns a geometry (not an optimized rectangle) that conforms to the rules of a geodetic geometry (each polygon element's area will be less than half the surface area of the Earth).
 - If <to_srid> is not a geodetic SRID, the optimized rectangle's SRID will be set to <to_srid> and returned.

ORACLE

8-18

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.VIEWPORT_TRANSFORM

As a convenience routine for visualizer applications, Oracle Spatial provides the SDO_CS.VIEWPORT_TRANSFORM function.

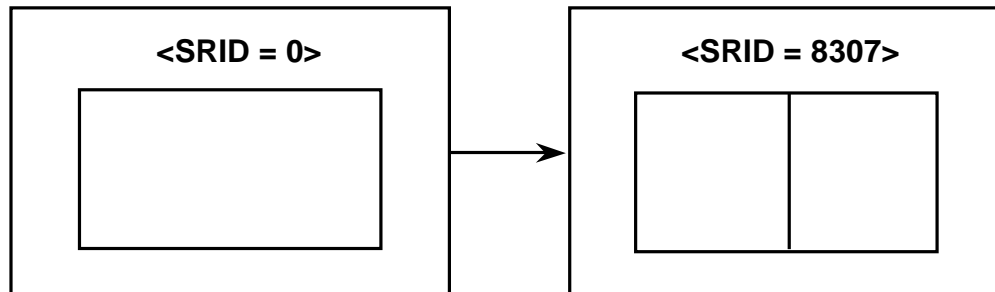
<geom> - must be an optimized rectangle with SRID set to 0.

<to_srid> - This is the SRID to transform the optimized rectangle to.

<returns> -

- If the <to_srid> is a geodetic SRID, returns a geometry that is not an optimized rectangle. The geometry returned will conform to the rules of a geodetic geometry (either a single polygon, or a multipolygon, where each of the the polygon element areas will be less than half the earth's surface area).
- If the <to_srid> is not a geodetic SRID, returns the same optimized rectangle with the SRID set to <to_srid>.

SDO_CS.VIEWPORT_TRANSFORM <to_srid> Is Geodetic



- If the <to_srid> is geodetic, the geometry returned may be a multipolygon
- If the <to_srid> is geodetic, only use the returned polygon in SDO_FILTER or SDO_RELATE with the ANYINTERACT mask

ORACLE

8-19

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.VIEWPORT_TRANSFORM When <to_srid> Is Geodetic

This is an example of passing in an optimized rectangle to SDO_CS.VIEWPORT_TRANSFORM, when the <to_srid> is geodetic.

Note that the returned geometry may generate a multipolygon made up of many polygons whose edges touch. This is done to ensure each of the polygon elements of the returned geometry will have 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 should only be used for the following operations:

- As a window for SDO_FILTER queries
- As a window for SDO_RELATE queries, but only for the ANYINTERACT mask

Explicit Viewport Transformation (Query Against a Geodetic Layer)

- **GEOD_STATES** is geodetic, viewport transformed to WGS84

```
SELECT a.state
FROM geod_states a
WHERE sdo_filter(
  a.geom,
  sdo_cs.viewport_transform(
    mdsys.sdo_geometry(2003, 0, NULL,
      mdsys.sdo_elem_info_array(1,1003,3),
      mdsys.sdo_ordinate_array(-90,-45,90,45)),
    8307),
  'querytype=window') = 'TRUE';
```

Note: All viewport transformations require a 0 SDO_SRID field set in source viewport geometry. The input geometry must be an optimized rectangle.

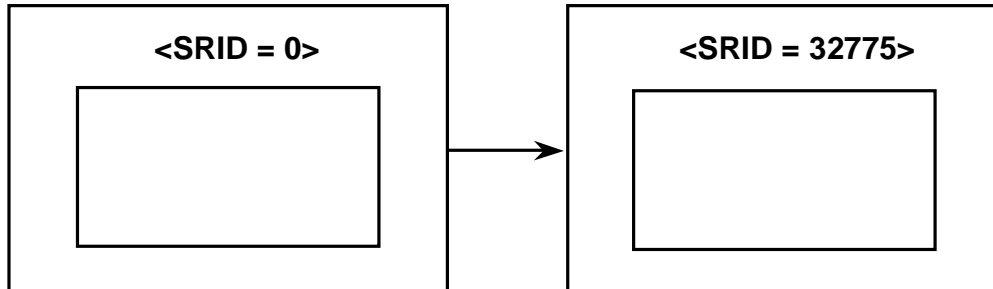
ORACLE

Explicit Viewport Transformation (Query Against a Geodetic Layer)

This is an example of an optimized rectangle query a visualizer application might use when querying a geodetic layer.

Because optimized rectangle windows are not valid windows for geodetic layers, the visualizer application must wrap the optimized rectangle in the `sdo_cs.viewport_transform` function.

SDO_CS.VIEWPORT_TRANSFORM <to_srid> Is Projected



- If the <to_srid> is projected, the function replaces the SRID of 0 with <to_srid>. The optimized rectangle is not modified
- If the <to_srid> is projected, the returned polygon can be used directly with SDO_FILTER or SDO_RELATE with any valid mask

ORACLE

8-21

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.VIEWPORT_TRANSFORM When <to_srid> Is Geodetic

This is an example of passing in an optimized rectangle to SDO_CS.VIEWPORT_TRANSFORM, when the <to_srid> is projected.

The returned geometry is the original optimized rectangle with its SRID set to <to_srid>.

The new geometry can be used in SDO_FILTER or SDO_RELATE with any valid mask.

Explicit Viewport Transformation (Query Against a Projected Layer)

- PROJ_STATES is projected (not latitude/longitude), viewport not really transformed, just SRID gets replaced

```
SELECT a.state
FROM proj_states a
WHERE sdo_filter(
  a.geom,
  sdo_cs.viewport_transform(
    mdsys.sdo_geometry(2003, 0, NULL,
      mdsys.sdo_elem_info_array(1,1003,3),
      mdsys.sdo_ordinate_array(-15156552, -5723852.4,
        15683827, 14911603.1)),
    32775),
  'querytype=window') = 'TRUE';
```

Note: All viewport transformations require a 0 SDO_SRID field set in source viewport geometry. The input geometry must be an optimized rectangle.

ORACLE

8-22

Copyright © Oracle Corporation, 2001. All rights reserved.

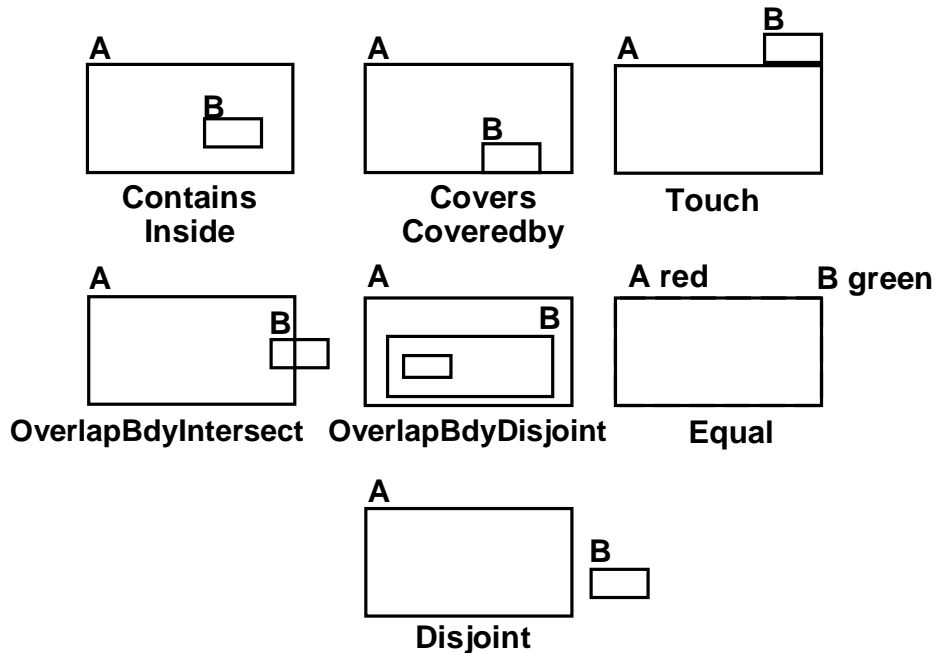
Explicit Viewport Transformation (Query Against a Projected Layer)

This is an example of an optimized rectangle query a visualizer application might use when querying a projected layer.

Even though optimized rectangle windows are valid windows for projected layers, a visualizer application may decide to wrap the optimized rectangle in the sdo_cs.viewport_transform function. This way optimized rectangles are treated the same way for both geodetic and projected layers.

When an optimized rectangle is sent to sdo_cs.viewport_transform, and the <to_srid> is a projected SRID (i.e. 32775), the sdo_cs.viewport_transform function will return the original optimized rectangle with the 0 SRID replaced with <to_srid>.

Spatial (Topological) Relationships



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: The opposite of CONTAINS. A INSIDE B implies B CONTAINS A.

COVERS: The interior of one object is completely contained in the interior of the other object and their boundaries intersect.

COVEREDBY: The opposite of COVERS. A COVEREDBY B implies B COVERS A.

TOUCH: The boundaries intersect but the interiors do not intersect.

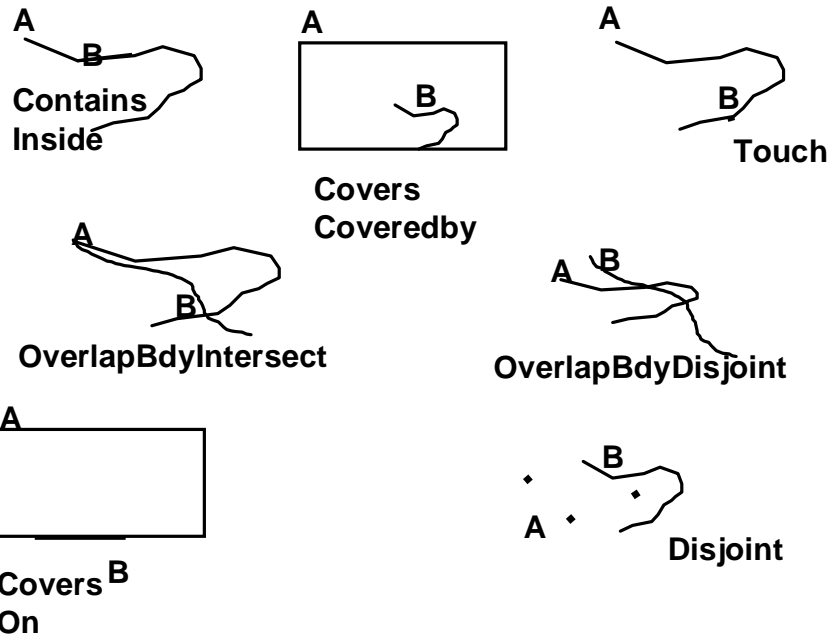
OVERLAPBDYINTERSECT: The boundaries and interiors of the two objects intersect.

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.

DISJOINT: The boundaries and interiors do not intersect.

Spatial (Topological) Relationships



ORACLE

8-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial (Topological) Relationships (continued)

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.

Spatial (Topological) Relationships

- **ANYINTERACT**
 - Returns TRUE if not disjoint
 - Optimal mask
 - In 8.1.6 was only mask to take advantage of center tile optimization

ORACLE

8-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial (Topological) Relationships (continued)

ANYINTERACT: The objects are non-disjoint.

This is an optimal mask that uses primary filter optimizations with both quadtree and R-tree indexes. It returns TRUE if not disjoint.

The SDO_RELATE Operator

```
SDO_RELATE  
( <geometry-1>,  
  <geometry-2>,  
  'MASK=<mask>  
    QUERYTYPE=<querytype>  
    [other optional parameters]'  
) = 'TRUE'
```

- Performs a primary and secondary filter
- Returns TRUE or FALSE

ORACLE

8-26

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Operator

SDO_RELATE uses the spatial index to identify either geometries that have a particular spatial interaction with an area of interest geometry, or pairs of geometries that have a particular spatial interaction. This operator performs both primary and secondary filter operations.

The template for SDO_RELATE is similar to SDO_FILTER. The first parameter, *geometry1*, is the column you are searching. The second parameter, *geometry2*, 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' .

Required Arguments

- **GEOMETRY-1**
 - Must be a column in a table
 - Must be of type **SDO_GEOMETRY**
 - Must be indexed
- **GEOMETRY-2**
 - Variable or column in a table
 - Must be of type **SDO_GEOMETRY**
 - If **querytype=JOIN** (very rare), it must be a column in a table and for best performance indexed with the same index type (quadtree/R-tree) and at the same **SDO_LEVEL** (if quadtree)

ORACLE

8-27

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Syntax

GEOMETRY-1

Specifies a geometry column in a table. The column must be spatially indexed and of the data type **MDSYS.SDO_GEOMETRY**. This is the spatial column being searched.

GEOMETRY-2

Specifies either a geometry from a table or a transient instance of a geometry (specified using a bind variable or **SDO_GEOMETRY** constructor). Its data type is **MDSYS.SDO_GEOMETRY**.

Required Arguments

- **MASK**

- Identify spatial relationship to test
- ORed masks are specified as follows (INSIDE+COVEREDBY).
- Oracle Spatial optimizes some ORed masks
 - INSIDE+COVEREDBY
 - CONTAINS+COVERS
 - In some very rare cases, when comparing geometries with lots of vertices, you can get better performance with multiple SDO_RELATE operations, each with one mask, separated by ORs.

ORACLE

8-28

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Syntax

MASK

Specifies the topological relation of interest. This is a required parameter. Valid values are one or more of the following in the 9-intersection pattern: TOUCH, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, EQUAL, INSIDE, COVEREDBY, CONTAINS, COVERS, ANYINTERACT, ON.

Oracle has optimized the combination of certain masks with the logical Boolean operator OR, for example, 'mask=inside+coveredby' or 'mask=contains+covers'.

In some very rare cases, when comparing geometries with high vertex counts, the following query may be more optimal. The following query only specifies one mask per SDO_RELATE. By specifying only one mask in SDO_RELATE, you ensure interior tile optimizations are invoked (that is, potentially accepting geometries without comparing their edges and interiors).

SDO_RELATE Syntax (continued)

But specifying more than one SDO_RELATE will invoke the primary filter operation multiple times. The tradeoff is how much time can you save with the interior tile optimization to overcome invoking the primary filter multiple times. Because in Oracle9i some major performance enhancements were made when comparing geometries with high vertex counts, the SQL statement below will only help in very rare cases.

For example:

```
select /*+ ordered */ count(*)
from geod_states a, geod_counties b
where a.state = 'New Hampshire'
      AND (sdo_relate (b.geom, a.geom, 'mask=TOUCH querytype=WINDOW') =
'TRUE' or
           sdo_relate (b.geom, a.geom, 'mask=INSIDE
querytype=WINDOW') = 'TRUE');
```

Required Arguments

- **QUERYTYPE**
 - Valid values are **WINDOW** or **JOIN** (very rare)

ORACLE

8-30

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Syntax

QUERYTYPE

Specifies valid query types: **WINDOW** or **JOIN**. This is a required parameter. **WINDOW** is recommended in almost all cases.

WINDOW implies that a comparison is performed between the index entries associated with *geometry2* (the query window or area-of-interest) and each index entry for the layer associated with *geometry1*.

The extreme example of a **WINDOW** constrains *geometry2* to a single geometry, but window queries do not limit *geometry2* to a single geometry.

JOIN is rarely used, and not recommended. **JOIN** compares all the geometry index records of a column to all the geometry index records of another column (you can think of this as a spatial cross product). **JOIN** requires spatial indexes exist on both parameters, *geometry1* and *geometry2*.

Any **JOIN** query can be rewritten as a **WINDOW** query, which will be demonstrated in an upcoming slide.

Optional Arguments

- **IDXTAB1**
 - Index table to associate with first geometry in operator
 - By default, the primary index table is used
- **IDXTAB2**
 - Index table to associate with the second geometry in operator
 - By default, the primary index table is used
 - Only supported if QUERYTYPE=JOIN

ORACLE

8-31

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Syntax

IDXTAB1

Specifies the name of the spatial index table to use for *geometry1*, if there are multiple spatial index tables for *geometry1*. By default, the primary spatial index table is used.

IDXTAB2

Specifies the name of the spatial index table to use for *geometry2*, if there are multiple spatial index tables for *geometry2*. By default, the primary spatial index table is used. IDXTAB2 is valid only if *querytype* is JOIN (which is not recommended).

Optional Arguments

- **LAYER_GTYPE** (deprecated in Oracle9i)
 - Before Oracle9i Spatial, set to **POINT** if querying **POINT** only columns (for optimal query performance)
 - Otherwise, **NOTPOINT** (default)
 - In Oracle9i Spatial and later, set the **LAYER_GTYPE** parameter in the **CREATE INDEX** command instead

ORACLE

8-32

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Syntax

The **LAYER_GTYPE** keyword has been deprecated in this operator, and it is ignored if specified.

Note: Specify the **LAYER_GTYPE** when creating a spatial index, as one of the **CREATE INDEX** parameters.

SDO_RELATE: A Window Query

- 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 querytype=WINDOW')='TRUE';
```

Note: For optimal performance, don't forget to index GEOD_STATES (state).

ORACLE

8-33

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Example

This query finds all the counties in the state of New Hampshire. It includes all the counties completely inside NH, as well as the counties inside NH that also touch its border.

Since the GEOD_COUNTIES table has a STATE_ABRV column, this query could have been done without Oracle Spatial. The query could have selected all the counties with state_abrv = 'NH'.

But to accomplish this query with Oracle Spatial, you can use SDO_RELATE.

SDO_RELATE performs a primary and a secondary filter and gives the exact result.

In the WHERE clause:

- geometry1 -- the search column -- is c.geom from the geod_counties table
- geometry2 -- the window -- is s.geom which corresponds to the state of New Hampshire
- mask=INSIDE+COVEREDBY that is, ORed mask
- querytype=WINDOW

SDO_RELATE: A Window Query

- 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 querytype=WINDOW') = 'TRUE';
```

ORACLE

8-34

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Example

This query finds all the counties that touch the border of Passaic, NJ. The SDO_RELATE uses a primary and a secondary filter and gives the exact result.

This query could not be done without Oracle Spatial. There are not attribute columns you can query without Oracle Spatial to satisfy the TOUCH relationship.

Note in this query that the GEOD_COUNTIES table is listed twice because the window (that is, Passaic county in New Jersey) is coming out of the same layer being searched.

In the WHERE clause:

- geometry1 -- the search column -- is C1.GEOM from the GEOD_COUNTIES table
- geometry2 -- the window -- is C2.GEOM which is Passaic county in New Jersey
- mask=TOUCH
- querytype=WINDOW

SDO_RELATE: A Window Query

- Find all counties that are **INSIDE** or **TOUCH** Texas

```
select c.county, c.state_abrv
from proj_counties c, proj_states s
where s.state = 'Texas'
and (sdo_relate (c.geom, s.geom,
                'mask=INSIDE querytype=WINDOW') = 'TRUE'
or
    sdo_relate (c.geom, s.geom,
                'mask=TOUCH querytype=WINDOW') = 'TRUE');
```

Hint 2: In some very rare cases, this query uses interior tile optimizations, and may be more optimal when the geometries you are comparing have lots of vertices.

ORACLE

8-35

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Example

Oracle has optimized the combination of certain masks with the logical Boolean operator OR, for example, 'mask=inside+coveredby' or 'mask=contains+covers'.

In some very rare cases, when comparing geometries with high vertex counts, the following query may be more optimal. The following query only specifies one mask per SDO_RELATE. By specifying only one mask in SDO_RELATE, you ensure interior tile optimizations are invoked (for example, potentially accepting geometries without comparing their edges and interiors).

But specifying more than one SDO_RELATE will invoke the primary filter operation multiple times. The tradeoff is how much time can you save with the interior tile optimization to overcome invoking the primary filter multiple times. Because some major performance enhancements were made when comparing geometries with high vertex counts in Oracle9i, the SQL statement below will only help in very rare cases.

For example:

```
select /*+ ordered */ count(*)
from geod_states a, geod_counties b
where a.state= 'New Hampshire'
AND (sdo_relate (b.geom, a.geom, 'mask=TOUCH querytype=WINDOW') =
'TRUE' or sdo_relate (b.geom, a.geom, 'mask=INSIDE
querytype=WINDOW') = 'TRUE');
```

SDO_RELATE: A Window Query

- Find all cities in a selected rectangular area

```
select c.city, c.pop90
from geod_cities c
where sdo_relate ( c.location,
                  sdo_cs.viewport_transform (
                    mdsys.sdo_geometry (2003, 0, null,
                    mdsys.sdo_elem_info_array (1,1003,3),
                    mdsys.sdo_ordinate_array (-109,37,-102,40)),
                    8307),
                  'mask=ANYINTERACT querytype=WINDOW')='TRUE';
```

Note: For point in polygon queries use the ANYINTERACT mask if you don't mind returning points which fall on the boundary; ANYINTERACT is a very fast relate operation.

ORACLE

8-36

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Example

This query finds all the cities in a selected rectangular area. The SDO_RELATE uses a primary and a secondary filter and gives the exact result.

In the WHERE clause:

- geometry1 -- the search column -- is C.LOCATION from the GEOD_CITIES table
- geometry2 -- the window -- is an optimized rectangle passed into viewport_transform.
- mask=ANYINTERACT
- querytype=WINDOW

Note

- For point in polygon queries use the ANYINTERACT mask if you don't mind returning points which fall on the boundary; ANYINTERACT is a very fast relate operation.
- SDO_CS.VIEWPORT_TRANSFORM is called here because optimized rectangles are not valid windows for geodetic layers. But SDO_RELATE should only use the result of SDO_CS.VIEWPORT_TRANSFORM as a window when the mask is ANYINTERACT (for example, the query in the slide).

SDO_RELATE and PL/SQL

- Find total population in a selected rectangular area

```
set serveroutput on;
declare
  rectangle mdsys.sdo_geometry;
  total_population number;
begin
  rectangle := sdo_cs.viewport_transform (
    mdsys.sdo_geometry (2003, 0, null,
      mdsys.sdo_elem_info_array (1,1003,3),
      mdsys.sdo_ordinate_array (-109,37,-102,40)),
    8307);
  select sum(c.totpop) into total_population
  from geod_counties c
  where sdo_relate (c.geom, rectangle,
    'mask=ANYINTERACT querytype=WINDOW') = 'TRUE';
  dbms_output.put_line('Population = ' || total_population || '.');
end;
/
```

ORACLE

8-37

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE PL/SQL Example

This query finds the total population of all counties that have any interaction with a selected rectangular area using PL/SQL. The SDO_RELATE operator uses a primary and a secondary filter and gives the exact result.

Note: SDO_CS.VIEWPORT_TRANSFORM is called here because optimized rectangles are not valid windows for geodetic layers. But SDO_RELATE should only use the result of SDO_CS.VIEWPORT_TRANSFORM as a window when the mask is ANYINTERACT (for example, the query in the slide).

SDO_RELATE: Window Query

- 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 querytype=WINDOW')='TRUE';
```

- 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 querytype=WINDOW')='TRUE';
```

Hint 3: When the query window comes from a table use the ordered optimizer hint and specify the query window (geometry-2) first in the from clause.

ORACLE

8-38

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Examples

The first query finds all the interstates that interact with the county Passaic.

In the WHERE clause:

- c.geom - corresponds to Passaic county in New Jersey (single window).
- geometry1 -- the search column -- I.GEOM from the GEOD_INTERSTATES table.
- geometry2 -- the window -- is C.GEOM which is the Passaic county in New Jersey.

The second query finds all the interstates that interact with counties where the population per square mile is less than 10 persons.

In the WHERE clause:

- c.geom - Corresponds to all the counties in Arizona with population/square mile < 10. This evaluates to more than one window. This is how you write a JOIN query as a WINDOW query.
- geometry1 -- the search column -- I.GEOM from the GEOD_INTERSTATES table.
- geometry2 -- the window -- is C.GEOM which are all the counties in Arizona with poppsqmi < 10.

Note

- 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 the `WINDOW` should always come first.
- The `/* +ordered */` hint must be used for optimal performance when multiple windows are passed into the second argument of a spatial operator (that is, `SDO_FILTER`, `SDO_RELATE`, `SDO_WITHIN_DISTANCE`, `SDO_NN`).

SDO_RELATE: Join Query

- Select all the city, county pairs

```
select city, county
from cities i, counties c
where sdo_relate (c.geom, i.location,
                  'mask=ANYINTERACT
                  querytype=JOIN') = 'TRUE';
```

Note:

- If you use querytype=JOIN, no interior tile optimization is done
- It may be better to rewrite this join query as a window query. But keep the following in mind (in order of importance):
 - Driving table should be the table with less rows. Driver geometries are specified in second argument of SDO_RELATE.
 - The second argument to relate is a polygon layer to take advantage of interior tile optimization.

ORACLE

8-40

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_RELATE Example

This query uses querytype=JOIN to find all the city, county pairs. The SDO_RELATE uses a primary and a secondary filter and gives the exact result.

In the WHERE clause:

- geometry1 -- the search column -- is c.geom from the counties table
- geometry2 -- the search column -- is i.location from the cities table
- mask=ANYINTERACT
- querytype=JOIN

Note: All JOIN queries can be rewritten as WINDOW queries. To rewrite a JOIN query as a WINDOW query:

1. Change the querytype=JOIN to querytype=WINDOW
2. Make geometry1 -- the search column -- the larger table (table with more rows)
3. Make geometry2 -- the window -- the smaller table (table with less rows)
4. Use the /*+ordered*/ hint

The rewritten version of the given query is:

```
select /*+ordered */ city, county
from cities i, counties c
where sdo_relate (c.geom, i.location, 'mask=ANYINTERACT
                                     querytype=window') = 'TRUE';
```

The SDO_WITHIN_DISTANCE Operator

```
SDO_WITHIN_DISTANCE  
( <geometry-1>,  
  <geometry-2>,  
  'DISTANCE=<n>,'  
    QUERYTYPE=<querytype>  
    [other optional parameters]'  
) = 'TRUE'
```

- Performs an exact or approximate query
- Returns TRUE or FALSE

ORACLE

8-41

Copyright © Oracle Corporation, 2001. All rights reserved.

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 of interest or point of interest). This operator performs an exact query by default, or can be made to perform an approximate query.

The expression SDO_WITHIN_DISTANCE(arg1, arg2, arg3) = 'TRUE' evaluates to TRUE for object pairs that are within the specified distance, and FALSE otherwise.

The template for SDO_WITHIN_DISTANCE is similar to SDO_FILTER. The first parameter, *geometry1*, is the column you are searching. The second parameter, *geometry2*, is the window (or area of interest) that you will specify a distance from. The third parameter is a quoted string of parameters specific to the operator.

The SDO_WITHIN_DISTANCE operator computes the distance between two geometries. The distance between two extended objects (non-point 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 geometry column must have a spatial index built on it. If the data is geodetic, the spatial index must be an R-tree index. The operator must always be used in a WHERE clause.

Arguments

- **GEOMETRY-1**
 - Must be a column in a table
 - Must be of type **SDO_GEOMETRY**
 - Must be indexed
- **GEOMETRY-2**
 - Variable or column in a table
 - Must be of type **SDO_GEOMETRY**
 - Will be buffered by distance
- **DISTANCE (required)**
 - The distance (expressed in the units used for the coordinate system, or by the **UNIT** parameter)

ORACLE

8-42

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_WITHIN_DISTANCE Syntax

GEOMETRY-1

Specifies a geometry column in a table. The column must be spatially indexed. The data type is **MDSYS.SDO_GEOMETRY**. This is the spatial column being searched.

GEOMETRY-2

Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or **SDO_GEOMETRY** constructor.) The data type is **MDSYS.SDO_GEOMETRY**. This is the geometry that will be buffered by **DISTANCE**.

DISTANCE

Specifies the distance value. If a coordinate system is associated with the geometry, the distance unit is assumed to be the unit associated with the coordinate system. This is a required parameter whose data type is **NUMBER**.

Note: If the **UNIT** parameter is not specified, the default unit for **DISTANCE** is the unit associated with the coordinate system, unless the layer you are querying is geodetic. If the layer you are querying is geodetic, and the **UNIT** parameter is not specified, the default unit for **DISTANCE** is meters.

Optional Arguments

- **QUERYTYPE (optional)**
 - If **FILTER**, primary filter query only (approximate results)
- **UNIT**
 - Unit of measure to associate with the **DISTANCE** parameter
 - If **GEOMETRY-2** is projected, and **UNIT** is not specified, defaults to the unit of measure associated with the coordinate system of **GEOMETRY-2**
 - If **GEOMETRY-2** is geodetic, and **UNIT** is not specified, default **UNIT** is **METER**

ORACLE

8-43

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_WITHIN_DISTANCE Syntax

QUERYTYPE

Set 'querytype = FILTER' to perform only a primary filter operation. If *querytype* is not specified, both primary and secondary filter operations are performed (default). The data type is VARCHAR2.

UNIT

UNIT specifies 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'). The default value for UNIT is the unit of measurement associated with the data for nongeodetic coordinate systems. For geodetic data (longitude/latitude), the default is METER.

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

8-44

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_WITHIN_DISTANCE Examples

The first query finds all the cities within 15 miles from an interstate, by default the result is exact (not a primary filter).

In the WHERE clause:

- I.HIGHWAY is I170 from the INTERSTATES table
- geometry1 -- the search column -- is C.LOCATION from the GEOD_CITIES table
- geometry2 -- the window -- is I.GEOM which is I170 from the GEOD_INTERSTATES table
- 'distance=15 unit=mile'

The second query finds all the interstates within 15 miles from a city.

In the WHERE clause:

- C.CITY is Tampa from the GEOD_CITIES table
- geometry1 -- the search column -- is I.GEOM from the GEOD_INTERSTATES table
- geometry2 -- the window -- is C.LOCATION which is Tampa
- 'distance=15 unit=mile'

The SDO_NN (Nearest Neighbor) Operator

```
SDO_NN  
( <geometry-1>,  
  <geometry-2>,  
  '[optional parameter]'  
) = 'TRUE'
```

- Returns the nearest neighbors to a geometry
- Can choose the number of nearest neighbors
- Results not returned in order of distance
- Can combine with other attributes in the WHERE clause
- Has an ancillary operator, SDO_NN_DISTANCE, that returns the distance associated with a nearest neighbor

ORACLE

8-45

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN

Determines the nearest neighbor geometries to a geometry. No assumptions should be made about the order of the returned results.

- If no optional parameters are specified, 1 nearest neighbor is returned.
- If you specify the optional SDO_NUM_RES parameter, you can request how many nearest neighbors you want, but no other conditions in the where clause are evaluated. For example, you want the five closest banks from an intersection, but only with bank name = 'CHASE'. If the five closest banks are not named 'CHASE', SDO_NN with SDO_BATCH_SIZE=5 will return no rows. SDO_NUM_RES only takes proximity into account.
- If you specify the optional SDO_BATCH_SIZE parameter, SDO_NN will keep returning neighbors in distance order to the WHERE clause (you can't assume order in the select list). If in your WHERE clause, you have the following "WHERE SDO_NN (...) = 'TRUE' and bank_name = 'CHASE' and rownum < 6", you can get back the five closest banks with bank_name = 'CHASE'.

SDO_NN_DISTANCE

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. SDO_NN_DISTANCE will be describe in an upcoming slide.

Arguments

- **GEOMETRY-1**
 - Must be a column in a table
 - Must be of type **SDO_GEOMETRY**
 - Must be indexed
 - This is the layer being searched for nearest neighbors
- **GEOMETRY-2**
 - Variable or column in a table
 - Must be of type **SDO_GEOMETRY**
 - This is the geometry whose nearest neighbors you are looking for

ORACLE

8-46

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN Syntax

GEOMETRY-1

Specifies a geometry column in a table. The column must be spatially indexed. The data type is **MDSYS.SDO_GEOMETRY**. This is the spatial column being searched for nearest neighbors.

GEOMETRY-2

Specifies either a geometry from a table or a transient instance of a geometry. (Specified using a bind variable or **SDO_GEOMETRY** constructor.) The data type is **MDSYS.SDO_GEOMETRY**. This is the geometry whose nearest neighbors you are looking for.

Arguments

- **SDO_NUM_RES (optional)**
 - The number of nearest neighbors to return
 - Default is 1 if neither SDO_NUM_RES nor SDO_BATCH_SIZE is specified
- **SDO_BATCH_SIZE (optional)**
 - When specified, SDO_NN will continually return neighbors in distance order so other attributes can be evaluated in the WHERE clause
 - Tuning parameter to instruct SDO_NN to compute <SDO_BATCH_SIZE> distances at a time
 - Requires an R-tree spatial index
- **Use either SDO_NUM_RES or SDO_BATCH_SIZE**

ORACLE

8-47

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN Syntax (continued)

SDO_NUM_RES (optional)

Specifies the number of nearest neighbors to return. SDO_NUM_RES only takes proximity into account, no assumptions should be made about the order of the returned results. When using the SDO_NUM_RES parameter, no other constraints are used in the WHERE clause.

SDO_BATCH_SIZE (optional)

The SDO_BATCH_SIZE number of rows is continuously returned until all the constraints in the WHERE clause are satisfied. This is a performance tuning parameter. You should choose a SDO_BATCH_SIZE that minimizes the number of distances SDO_NN will calculate internally to satisfy your query. The functionality associated with the SDO_BATCH_SIZE parameter is only available with an R-tree spatial index.

The use of SDO_NUM_RES and SDO_BATCH_SIZE is mutually exclusive. If neither is specified then the default is SDO_NUM_RES=1

Arguments

- **UNIT (optional with SDO_NN_DISTANCE)**
 - Applied to SDO_NN_DISTANCE (next slide)
 - Unit of measure to associate with distance
- **TAG (required with SDO_NN_DISTANCE)**
 - Match with integer tag specified in SDO_NN_DISTANCE
 - Associates ancillary operator with specific instance of SDO_NN

ORACLE

8-48

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN Syntax (continued)

The following SDO_NN parameters are associated with the SDO_NN_DISTANCE ancillary operator, which is discussed in the next slide.

UNIT (optional)

Specifies the unit of measure for the results returned by the SDO_NN_DISTANCE ancillary operator. SDO_NN_DISTANCE will be discussed in an upcoming slide. The default unit is the unit of measure associated with the data. For longitude/latitude data, the default 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.

SDO_NN_DISTANCE

Nearest Neighbor Ancillary Operator

SDO_NN_DISTANCE (<TAG>) returns NUMBER

- Returns the distance associated with nearest neighbors returned by SDO_NN
- Distance unit conforms to the UNIT parameter. If UNIT was not specified, SDO_NN_DISTANCE will default to the following units:
 - For geodetic data, default unit is meters
 - For projected data, default unit is the coordinate system unit
- TAG associates the ancillary operator with call to SDO_NN

ORACLE

8-49

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN_DISTANCE

Returns the distance associated with nearest neighbors returned by SDO_NN. The distance unit will conform to the UNIT parameter of SDO_NN. If UNIT was not specified in SDO_NN, SDO_NN_DISTANCE will default to the following units:

- for geodetic data, default unit is meters
- for projected data, default unit is the coordinate system unit

<TAG> This parameter is type NUMBER. There is no special meaning to this parameter. It is a value that must match the last parameter of SDO_NN (that is, it associates SDO_NN_DISTANCE with the SDO_NN operator).

SDO_NN Example

- Find the five cities nearest to interstate I170, ordered by distance

```
SELECT /*+ ordered */
       c.city,
       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;
```

Note: Make sure you have an index on GEOD_INTERSTATES (HIGHWAY).

ORACLE

8-50

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN Example

This query finds the five cities nearest to interstate I170, and returns the distance, ordered by distance in miles.

Because the `/*+ ordered */` hint was used, it is important to have an index on the `geod_interstates(highway)` column, and make that the query driver. This will force the query to locate highway 'I170' before it tries to find nearest neighbors.

In the WHERE clause:

- `I.HIGHWAY` is I170 from the `GEOD_INTERSTATES` table
- `geometry1` -- the search column -- is `C.LOCATION` from the `GEOD_CITIES` table
- `geometry2` -- the window -- is `I.GEOM`, which is highway 'I170'
- `'sdo_num_res=5'` - how many nearest neighbors to find
- `'unit=mile'` - UNIT to associate with distances returned by `SDO_NN_DISTANCE`
- `1` - the tag to associate this call to `SDO_NN` to the call to `SDO_NN_DISTANCE`

The query does an `ORDER BY` to order the results by distance.

SDO_NN Example: Results

- Find the five cities nearest to Interstate I170, ordered by distance

| CITY | DISTANCE_IN_MILES |
|-------------|-------------------|
| ----- | ----- |
| St Louis | 5.36297295 |
| Springfield | 78.7997464 |
| Peoria | 141.478022 |
| Evansville | 158.22422 |
| Springfield | 188.508631 |

ORACLE

8-51

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN Example Results

The five closest cities are returned, ordered by distance in miles. Nearest neighbor (SDO_NN) was the only constraint associated with the spatial data returned in the WHERE clause.

SDO_NN Example

- Find the five cities nearest to Interstate I170 that have population greater than 300000, ordered by distance

```
SELECT /*+ ordered */
       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=10 unit=mile', 1) = 'TRUE'
       AND c.pop90 > 300000
       AND rownum < 6
ORDER BY distance_in_miles;
```

Note: Make sure you have an index on GEOD_INTERSTATES (HIGHWAY).

Note: SDO_NN returns rows in distance order to the WHERE clause, so it is OK to have the rownum stopping condition followed by an ORDER BY (traditionally this is not allowed).

ORACLE

SDO_NN Example

This query finds the five cities nearest to interstate I170 that have population greater than 300,000 people, and returns the city, the population in 1990, and the distance, ordered by distance in miles.

The SDO_BATCH_SIZE parameter will continually return closest neighbors (in distance order) to the other conditions in the WHERE clause.

Because the /*+ ordered */ hint was used, it is important to have an index on the geod_interstates(highway) column, and make that the query driver. This will force the query to locate highway 'I170' before it tries to find nearest neighbors.

SDO_NN Example (continued)

In the WHERE clause:

i.highway -- I170 from the geod_interstates table

geometry1 -- the search column -- is c.location from the geod_cities table

geometry2 -- the window -- is i.geom which is highway 'I170'

'sdo_batch_size=10' -- How many nearest neighbor distances to calculate at a time. Since the five closest cities might not start with 'S', this parameter tells SDO_NN to calculate 10 distances at a time until all the other conditions in the WHERE clause are met.

'unit=mile' - UNIT to associate with distances returned by SDO_NN_DISTANCE

1 - This is the tag to associate this call to SDO_NN to the call to SDO_NN_DISTANCE

AND pop90 > 300000 - This is the additional clause that specifies the population in the city

AND rownum < 6 - Stopping condition for the SQL statement. Stop when there are five candidates that meet all the conditions in the WHERE clause.

The query does an ORDER BY to order the results by distance. Normally, a rownum predicate would 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 could order 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.

SDO_NN Example: Results

- Find the five cities nearest to Interstate I170 that have population greater than 300000, ordered by distance

| CITY | POP90 | DISTANCE_IN_MILES |
|--------------|---------|-------------------|
| ----- | ----- | ----- |
| St Louis | 396685 | 5.36297295 |
| Kansas City | 435146 | 227.404883 |
| Indianapolis | 741952 | 234.708666 |
| Memphis | 610337 | 244.202072 |
| Chicago | 2783726 | 253.547961 |

ORACLE

8-54

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_NN Example Results

The five closest cities are returned that have a population of over 300,000 people, ordered by distance in miles. Nearest neighbor (SDO_NN), with the SDO_BATCH_SIZE parameter, enabled nearest neighbors to continually return until all the other WHERE clause conditions were met.

The SDO_GEOM.RELATE Function

```
SDO_GEOM.RELATE
( <geometry-1>, <diminfo-1>,
  '<mask>',
  <geometry-2>, <diminfo-2> )= '<relationship>'
or
SDO_GEOM.RELATE
( <geometry-1>, '<mask>', <geometry-2>, <tolerance> )
= '<relationship>'
```

- Performs an exact query (secondary filter)
- Returns TRUE or FALSE for an ANYINTERACT mask
- Returns the matching relationship if any other mask or FALSE
- Can be used in the select list

ORACLE

8-55

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.RELATE Function

This function (and all spatial functions) does not use a spatial index. Examines two geometries to determine their spatial relationship. This function is almost never used in the WHERE clause (SDO_RELATE is used instead). 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.

The DETERMINE mask to find out the relationship between two geometries.

SDO_GEOM.RELATE Parameters

- **GEOMETRY1**
- **DIMINFO1**
 - From user_sdo_geom_metadata table
- **MASK**
 - Mask for operation
- **GEOMETRY2**
- **DIMINFO2**
 - From user_sdo_geom_metadata table
- **TOLERANCE**
 - The tolerance value to be used
- **RELATIONSHIP**
 - Returned value
 - **TRUE** for **ANYINTERACT**, or **MASK** if matches
 - Otherwise, **FALSE**

ORACLE

8-56

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.RELATE Syntax

GEOMETRY1

- Geometry object.

DIMINFO1

- Dimensional information array corresponding to geometry1, usually selected from one of the xxx_SDO_GEOM_METADATA views

MASK

- Specifies a list of relationships to check.

GEOMETRY2

- Geometry object.

DIMINFO2

- Dimensional information array corresponding to geometry2, usually selected from one of the xxx_SDO_GEOM_METADATA views

TOLERANCE

- Tolerance value

RELATIONSHIP

- The result (returned value) of the query that contains the MASK if success (or TRUE if ANYINTERACT was specified), or FALSE if the specified relationship was not matched.

Note: If your data is in the four digit SDO_GTYPE/SDO_ETYPE, the TOLERANCE version of this function is shorter and easier to read.

SDO_GEOM.RELATE Function

- Determine 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;
```

| COUNTY | RELATIONSHIP |
|------------|--------------|
| Atlantic | COVERS |
| Cape May | COVERS |
| Cumberland | COVERS |
| Essex | CONTAINS |

ORACLE

8-57

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.RELATE Example

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 Function

- Determine relationship of New Jersey's counties to the state of New Jersey (reverse relationship of previous slide).

```
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;
```

| COUNTY | RELATIONSHIP |
|------------|--------------|
| ----- | ----- |
| Atlantic | COVEREDBY |
| Cape May | COVEREDBY |
| Cumberland | COVEREDBY |
| Essex | INSIDE |

ORACLE

8-58

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.RELATE Example

This example determines the relationship of New Jersey's counties to the state of New Jersey. This is the inverse relationship from 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 Function

- 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 spatial indexes !
 - Use SDO_RELATE instead.

ORACLE

8-59

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.RELATE Function

This query finds all the counties around the state of New Jersey, but does not use a spatial index. SDO_GEOM.RELATE is almost never called in the WHERE clause of a SQL statement because it doesn't take advantage of spatial indexes. Use SDO_RELATE instead.

Oracle 9i Intermedia Locator

- **Bundled in Oracle9i**
- **Includes all the spatial operators**
 - **SDO_FILTER**
 - **SDO_RELATE**
 - **SDO_WITHIN_DISTANCE**
 - **SDO_NN and SDO_NN_DISTANCE**
- **Includes the SDO_GEOM.SDO_DISTANCE function (discussed in the next lesson)**
- **Can operate on all the supported element types (prior to Oracle9i, was limited to point data)**
- **Includes VIEWPORT_TRANSFORM**
- **Includes VALIDATE_GEOMETRY and VALIDATE_LAYER**
- **Includes SDO_MIGRATE.TO_CURRENT**
- **Can perform implicit transformations**
- **Is available on the Oracle Standard Edition**

ORACLE

8-60

Copyright © Oracle Corporation, 2001. All rights reserved.

LOCATOR_WITHIN_DISTANCE Example

As of Oracle9i, interMedia Locator includes all the spatial operators:

- SDO_FILTER
- SDO_RELATE
- SDO_WITHIN_DISTANCE
- SDO_NN and SDO_NN_DISTANCE

interMedia Locator also includes the SDO_GEOM.SDO_DISTANCE function. This helps you get the distances associated with geometries that return from SDO_WITHIN_DISTANCE.

As of Oracle9i, interMedia Locator supports all the element types supported by Oracle Spatial. Prior to Oracle9i, only point data was supported for interMedia Locator.

As of Oracle9i, interMedia Locator is available on Oracle Standard Edition as well as Oracle Enterprise Edition.

Summary

In this lesson, you should have learned how to:

- **Describe a spatial window query and a spatial join**
- **Explain the Oracle Spatial query model**
- **Describe and compare spatial operators and functions**
- **Use the spatial operators and functions to perform spatial queries**
- **Describe the topological relationships used by the spatial operators**

ORACLE

