
Oracle9i: Spatial

Student Guide
Volume 2

D12848GC10
Edition 1.0
November 2001
D34081

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

9

Spatial Analysis

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Use the advanced spatial functions to perform complex analyses**
- **Perform length and area calculations**
- **Perform distance calculations**
- **Generate buffers around geometries**
- **Perform geometry manipulations**
- **Use spatial aggregate functions**

ORACLE

9-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

This lesson describes the process of building and executing advanced spatial queries.

Area and Length Calculations

- **SDO_GEOM.SDO_AREA**
 - Computes the area of a polygon
- **SDO_GEOM.SDO_LENGTH**
 - Computes the length or perimeter of a geometry
- Both functions take an **SDO_GEOMETRY** object as input and return a number

ORACLE

9-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Area and Length Calculations

Oracle Spatial provides specialized functions for determining the area and length of geometries.

`SDO_GEOM.SDO_AREA` computes the area of a polygon.

`SDO_GEOM.SDO_LENGTH` computes the length of a line string, or the length of the perimeter of a polygon. If the polygon has holes (voids), then the length is the length of the outside perimeter of the polygon in addition to the length of the perimeter of the voids.

The input to both functions is an `SDO_GEOMETRY` object, which can come from a table, from a variable, or from a constructor.

Both functions return a number.

Area and Length Calculations

- **Projected data, or data with no SDO_SRID**
 - «Cartesian» calculations
 - Results default to the coordinate system unit (area = units²)
- **Geodetic data**
 - Calculations account for curvature of the earth
 - Results default to meter or meter²
- **Result unit can be specified with the UNIT parameter**

ORACLE

9-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Area and Length Calculations (continued)

Before Oracle9i Spatial, all calculations using these functions were Cartesian in nature. Now calculations are Cartesian for projected data or data with a NULL SDO_SRID. Geodetic calculations can now be performed on geodetic data, resulting in correct answers that take into account the curvature of the earth.

For projected data, results of area and length calculations default to the units used to define the coordinate system (area results are in units²)

For geodetic data, results default to meters (for length) and meters² (for area).

The default units specification can be overridden by including the unit parameter.

The SDO_AREA Function

```
area := SDO_GEOM.SDO_AREA (<geometry>,  
                             <diminfo> [, <unit>])  
or  
area := SDO_GEOM.SDO_AREA (<geometry>,  
                             <tolerance> [, <unit>])
```

- **<geometry> = SDO_GEOMETRY that defines a polygon**
 - Can be a variable or table column
 - Can be the result of another function
- **<diminfo> = dimension array**
- **<tolerance> = number used as the tolerance**
- **<unit> = a quoted string with the units for the result**
- **returned value = a number**

ORACLE

9-5

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_GEOM.SDO_AREA Function

The SDO_GEOM.SDO_AREA function is a function that calculates the area of a polygon. It is an overloaded function that has two distinct footprints. Both require the first parameter to be an object of type SDO_GEOMETRY. This object can come from a table, a variable or a constructor, or it can be the result of another function that returns a SDO_GEOMETRY object.

The second parameter changes based on which footprint is used. The first footprint requires that the diminfo array be passed into the function for the geometry object. This is the *required version* of the SDO_GEOM.SDO_AREA function call *if the data has not been migrated to include four digit SDO_GTYPE values* which include the dimensionality.

If the geometry objects have been specified with four digit SDO_GTYPE values, then the tolerance footprint of the function can be used.

Both footprints optionally allow a third parameter that specifies the UNIT for the results. This is formatted within a quoted string, for example, 'UNIT=ACRE' or 'UNIT=SQ_MILE'. There are 48 different values allowed for the UNIT area parameter, although some values have the same meaning (for example, SQ_KM and SQ_KILOMETER). A list of supported values for the UNIT parameter is available by doing the following query:

```
select sdo_unit from mdsys.sdo_area_units;
```

SDO_AREA Examples

- Calculate the total area of all counties around Passaic County

```
SELECT SUM(SDO_GEOM.SDO_AREA (c1.geom, 0.5, 'unit=sq_mile'))
       area
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';

      AREA
-----
2431.49227
```

ORACLE

9-6

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_AREA Examples

This example shows how to calculate the total area of all of the counties around Passaic County in New Jersey.

The WHERE clause specifies an SDO_RELATE operator where the query window is Passaic County in New Jersey, and to return to the SELECT clause all of the geometries for counties that have a TOUCH relationship to Passaic County.

The SELECT clause is using the SUM aggregate combined with the SDO_AREA function to accumulate the area of all of the counties returned from the where clause.

The result of this statement is a single value which is the area in square miles of all of the counties that have a touch relationship with Passaic County in New Jersey.

SDO_AREA Examples

- Calculate the total area of counties around Passaic County, group by state

```
SELECT c1.state,  
       SUM(SDO_GEOM.SDO_AREA(c1.geom,0.5,'unit=sq_mile'))  
       area  
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'  
GROUP BY c1.state;
```

STATE	AREA
-----	-----
New Jersey	1392.4091
New York	1039.08317

ORACLE

9-7

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_AREA Examples

This example shows how to calculate the total area of all of the counties around Passaic County in New Jersey, using a GROUP BY clause to separate the results by state.

The WHERE clause specifies an SDO_RELATE operator where the query window is Passaic County in New Jersey, and to return to the SELECT clause all of the geometries for counties that have a TOUCH relationship to Passaic County.

The SELECT clause is using the SUM aggregate combined with the SDO_AREA function to accumulate the area of all of the counties returned from the where clause. In addition, due to the group by clause, the aggregation is done separately for each state that contains counties that touch Passaic County.

The result of this statement is the state name and single value for each state which is the area in square miles of all of the counties in that state that have a touch relationship with Passaic County in New Jersey.

The SDO_LENGTH Function

```
length := SDO_GEOM.SDO_LENGTH
( <geometry>, <diminfo> [, <unit>])
or
length := SDO_GEOM.SDO_LENGTH
( <geometry>, <tolerance> [, <unit>])
```

- **<geometry> = SDO_GEOMETRY that defines a polygon or line**
 - Can be a variable or table column
 - Can be the result of another function
- **<diminfo> = dimension array**
- **<tolerance> = number used as the tolerance**
- **<unit> = a quoted string with the units for the result**
- **returned value = a number**

ORACLE

9-8

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_GEOM.SDO_LENGTH Function

The SDO_GEOM.SDO_LENGTH function is an overloaded function that calculates the length of a line segment, or the length of the perimeter of a polygon.

SDO_GEOM.SDO_LENGTH has two distinct footprints. Both require the first parameter to be an object of type SDO_GEOMETRY. This object can come from a table, a variable or a constructor, or it can be the result of another function that returns a SDO_GEOMETRY object.

The second parameter changes based on which footprint is used. The first footprint requires that the diminfo array be passed into the function for the geometry object. This is the *required version* of the SDO_GEOM.SDO_LENGTH function call *if the data has not been migrated to include 4 digit SDO_GTYPE values* which include the dimensionality.

If the geometry objects have been specified with four digit SDO_GTYPE values, then the tolerance footprint of the function can be used.

Both footprints optionally allow a third parameter that specifies the UNIT for the results. This is formatted within a quoted string, for example, 'UNIT=FOOT' or 'UNIT=METER'. There are 54 different values allowed for the UNIT distance parameter, although some values have the same meaning (for example, KM and KILOMETER). A list of supported values for the UNIT parameter is available by doing the following query:

```
select sdo_unit from mdsys.sdo_dist_units;
```

SDO_LENGTH Examples

- Calculate the length of an interstate highway

```
SELECT highway,  
       SDO_GEOM.SDO_LENGTH (geom, 0.5,'unit=kilometer')  
       length  
FROM geod_interstates  
WHERE highway = 'I95';
```

- Select states based on border length

```
SELECT state  
FROM geod_states  
WHERE SDO_GEOM.SDO_LENGTH (geom, 0.5,'unit=mile') > 1800;
```

ORACLE

SDO_GEOM.SDO_LENGTH Examples

The first example calculates the length of Interstate 95, returning the highway name (I95) and the length in kilometers. The where clause specifies 'I95', and the geometry column associated with I95 (geom) is the geometry that the SDO_GEOM.SDO_LENGTH function works on.

The second example applies the SDO_GEOM.SDO_LENGTH function to all of the geometries in the states layer (as calculated in the where clause), but only returns the states whose perimeter length is greater than 1800 miles.

Note that this lesson describes functions, and as mentioned previously, functions can be applied either in the select list or in the where clause. These examples show the use of functions in both places.

Distance Calculations

- **SDO_GEOM.SDO_DISTANCE** computes the minimum distance between two geometries
- This function takes two **SDO_GEOMETRY** objects as input and returns a number
- **Projected data (or data with no SRID)**
 - Returns straight line distance
 - Distance unit defaults to the coordinate system unit of the geometry
- **Geodetic data**
 - Returns great circle distance
 - Distance unit defaults to meters
- Distance unit can be specified with the **UNIT** parameter

ORACLE

9-10

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_GEOM.SDO_DISTANCE Function

The SDO_GEOM.SDO_DISTANCE function is a function that calculates the minimum distance between any two geometry objects. The function work with all types of geometries (the SDO_GEOM.SDO_AREA function works only with polygons, and SDO_GEOM.SDO_LENGTH function works with lines or polygons).

The SDO_GEOM.SDO_DISTANCE function takes two geometries as input, calculates the minimum distance between them, and returns that distance.

The distance returned is the straight line distance for projected data or data with no coordinate system associated with it, or it is the minimum great circle distance for geodetic data.

The default units for the SDO_GEOM.SDO_DISTANCE function are those in which the data was specified except for geodetic data, where the default units are meters. Users can override the default units.

The SDO_DISTANCE Function

```
distance := SDO_GEOM.SDO_DISTANCE
    ( <geometry-1>, <diminfo-1>,
      <geometry-2>, <diminfo-2> [, <unit>])
or
distance := SDO_GEOM.SDO_DISTANCE
    ( <geom1>, <geom2>, <tolerance> [, <unit>])
```

- **<geometry-n> = SDO_GEOMETRY objects**
 - Can be variables and/or table columns
 - Can be the result of another function
- **<diminfo-n> = dimension arrays**
- **<tolerance> = numeric tolerance for the function**
- **<unit> = a quoted string with the units for the result**
- **Returned value = a number (the minimum distance between the geometries)**

ORACLE

9-11

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_GEOM.SDO_DISTANCE Function

The SDO_GEOM.SDO_DISTANCE function calculates the minimum distance between any two geometry objects. SDO_GEOM.SDO_DISTANCE is an overloaded function; it has two distinct footprints. Both require two parameters of type SDO_GEOMETRY. These can come from a table, a variable or a constructor, or they can be the result of other functions that return an SDO_GEOMETRY object.

The use of the diminfo field or the tolerance changes based on which footprint is used. The first footprint requires that the diminfo array be passed into the function for the geometry object. This is the *required version* of the SDO_GEOM.SDO_DISTANCE function call *if the data has not been migrated to include four digit SDO_GTYPE values* which include the dimensionality.

If the geometry objects have been specified with four digit SDO_GTYPE values, then the tolerance footprint of the function can be used.

Both footprints optionally allow a parameter that specifies the UNIT for the results. This is formatted within a quoted string, for example, 'UNIT=FOOT' or 'UNIT=METER'. There are 54 different values allowed for the UNIT distance parameter, although some values have the same meaning (for example, KM and KILOMETER). A list of supported values for the UNIT parameter is available by doing the following query:

```
select sdo_unit from mdsys.sdo_dist_units;
```

SDO_DISTANCE Examples

- Distance from Buffalo to Syracuse in meters

```
SELECT SDO_GEOM.SDO_DISTANCE (
      a.location, b.location, 0.5) distance
FROM geod_cities a, geod_cities b
WHERE a.city = 'Buffalo' and b.city = 'Syracuse';
```

- Minimum distance between I84 and Passaic County in miles

```
SELECT SDO_GEOM.SDO_DISTANCE (
      a.geom, b.geom, 0.5, 'unit=mile') distance
FROM geod_interstates a, geod_counties b
WHERE b.county = 'Passaic' and b.state_abrv = 'NJ'
      AND a.highway = 'I84';
```

ORACLE

9-12

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_DISTANCE Examples

The first SDO_GEOM.SDO_DISTANCE example calculates the distance between the point location representing the city of Buffalo, NY and the point location of Syracuse, NY. The distance is calculated using the minimum great circle distance between the two points. The query is selecting from the CITIES table, sending the geometry object in the location fields of cities where the city is Buffalo in one instance, and Syracuse in the other instance, as the input geometries to the SDO_DISTANCE function. Since the units field hasn't been specified in this case, and the data is geodetic, the distance is in meters.

The second example determines the minimum distance between the highway I84 and Passaic County in New Jersey. The query is sending the geometry object stored in the geom column of the interstates table where the highway name is I84 as the first geometry in the distance function, and the second geometry is the geom column from the counties table where the county is Passaic County in New Jersey. The SDO_DISTANCE function returns the distance value in miles.

SDO_DISTANCE Examples

- Find all the cities within 100 miles of Boston, ordered by distance

```
SELECT a.city,  
       SDO_GEOM.SDO_DISTANCE (a.location, b.location,  
                              0.5, 'unit=mile') distance  
FROM geod_cities a, geod_cities b  
WHERE b.city = 'Boston'  
      AND SDO_WITHIN_DISTANCE(a.location, b.location,  
                              'distance=100 unit=mile') = 'TRUE'  
ORDER BY distance;
```

CITY	DISTANCE
Boston	0
Lowell	26.039707
Worcester	40.7874355
Providence	41.0584302
Springfield	79.5090887
Hartford	94.303397

ORACLE

SDO_GEOM.SDO_DISTANCE Examples (continued)

Note there is no ancillary operator for SDO_WITHIN_DISTANCE to return the distance associated with the results of the SDO_WITHIN_DISTANCE operator. In order to get the distance associated with the results of an SDO_WITHIN_DISTANCE operator a query can be written as shown in this example.

This query uses SDO_WITHIN_DISTANCE to return the set of cities within 100 miles of Boston, and then the function SDO_GEOM.SDO_DISTANCE is invoked for each of the cities returned. The ORDER BY clause puts the results in distance order.

Accuracy of Geodetic Area Calculations

Area calculation examples

- **Areas the size of New Hampshire: within 0.0001%**
 - Area of New Hampshire is 24042.6439 +/- 0.024 km²
- **Areas the size of India (approx. 1/3 size of US): within 0.001%**
 - Area of India is approximately 3277093.11 +/- 32.77 km²
- **Half the area of the Earth (biggest polygon): within 0.1%**
 - Approximately 255,082,311 +/- 255,082 km²
- **Larger differences in latitude introduce larger error margins, not to exceed the values listed**

ORACLE

9-14

Copyright © Oracle Corporation, 2001. All rights reserved.

Accuracy of Geodetic Area Calculations

The accuracy of the geodetic functions supplied with Oracle Spatial vary.

Area

- The accuracy of area calculations depends on geometry size and the how much it varies in Latitude. Larger differences in latitude introduce larger error margins, not to exceed the error margins listed.
- Smaller sized areas (for example New Hampshire) are accurate to within 0.0001% (about 1 part in a million). For New Hampshire the area is approximately 24,042.6439 square kilometers +/- 0.024 square kilometers.
- Larger sized areas (for example India), are accurate to within 0.001% (about 1 part in 100,000). For India the area is approximately 3277093.11 square kilometers +/- 32.77 square kilometers.
- An area covering 1/2 the Earth's surface is accurate to within 0.1%. For 1/2 the Earth's surface the area is approximately 255,082,311 square kilometers +/- 255,082 square kilometers.

Accuracy of Geodetic Length and Distance Calculations

- **Length calculations**
 - Generalized algorithm for short or long lengths
 - Accurate to within 0.00000001%
- **Distance calculations**
 - Point to point distances are same as Length, above
 - If not point to point, the algorithm is accurate to within 0.2%, and is generalized for short or long distances

ORACLE

9-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Accuracy of Geodetic Functions

The following describes the accuracy of the geodetic length and distance functions supplied with Oracle Spatial:

- **Length**
 - The algorithm for determining length values has been generalized for short or long lengths, and is accurate to 0.00000001% (or 1 part in 10 billion).
- **Distance**
 - Distance accuracy is dependent on the geometry types. Point to point distances are accurate to the same degree as length calculations. Other distance calculations are generalized for long or short distances, and are accurate to within 0.2% (or 2 parts per thousand).

Arc Densification and Buffering

ORACLE

9-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Arc Densification

- **SDO_GEOM.SDO_ARC_DENSIFY** comes with Oracle9i Spatial
- Takes a geometry as input that may have arcs, and returns a geometry that contains no arcs
- The spatial function **SDO_BUFFER** (discussed in an upcoming slide) implicitly densifies arcs
- Important for **SDO_BUFFER** to use this function when it buffers geodetic geometries, because the output geometry cannot have arcs or circles
- You may never have to call this function explicitly

ORACLE

9-17

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_ARC_DENSIFY

The **SDO_GEOM.SDO_ARC_DENSIFY** function is included in Oracle9i.

It takes an input geometry and densifies any circles or circular arcs into line segments composed of straight lines.

The spatial function **SDO_GEOM.SDO_BUFFER** will implicitly densify circular arcs when the geometry to be buffered is in a geodetic coordinate system.

Note that circular arcs and circles are not allowed in geodetic coordinate systems.

This function can be called explicitly, but more commonly will be implicitly called when **SDO_GEOM.SDO_BUFFER** works with geodetic data.

Arc Densification

- **SDO_GEOM.SDO_ARC_DENSIFY**
 - Creates a geometry with no circles or circular arcs
 - Takes an SDO_GEOMETRY object as input
 - Any geometry (including compound)
 - Can be the result of another function
 - Returns an SDO_GEOMETRY object containing no circles or circular arcs
 - Creates geometries that can be used in geodetic space

ORACLE

9-18

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_ARC_DENSIFY (continued)

The SDO_GEOM.SDO_ARC_DENSIFY function takes an input geometry and densifies any circles or circular arcs into line segments composed of straight lines.

The input geometry can be any type of SDO_GEOMETRY object supported by Oracle Spatial, including point, line, polygon, multipoint, multiline, multipolygon, compound line string or compound polygon. Polygons with holes are supported.

The geometry object can come from a table, it can be a variable or a constructor, or it can be the geometry resulting from another spatial function.

Any arc elements or circle elements within the geometries are densified or smoothed to a number of line segments such that the distance between the original arc and the line segment that approximates it never exceeds an arc tolerance value specified by the user.

The SDO_ARC_DENSIFY Function

```
SDO_GEOMETRY := SDO_GEOM.SDO_ARC_DENSIFY  
    ( <geometry>, <diminfo>, <params> )  
or  
SDO_GEOMETRY := SDO_GEOM.SDO_ARC_DENSIFY  
    ( <geometry>, <tolerance>, <params> )
```

- **<geometry> = SDO_GEOMETRY object**
 - can be a variable or table column
- **<diminfo> = dimension array**
- **<tolerance> = numeric tolerance of data**
- **<params> =**
 - **<arc-tolerance> (required)** - Maximum distance between arc and line approximating arc
 - **<unit> (optional)** - unit for <arc-tolerance>
- **returns = an SDO_GEOMETRY object with no arcs/circles**

ORACLE

9-19

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_ARC_DENSIFY Function

The SDO_GEOM.SDO_ARC_DENSIFY function takes an input geometry and densifies any circles or circular arcs into line segments composed of straight lines.

SDO_GEOM.SDO_ARC_DENSIFY is an overloaded function with two different footprints.

Both footprints require a parameter of type MDSYS.SDO_GEOMETRY. This parameter is the geometry to be buffered. It can come from a table, a variable or a constructor, or it can be the result of a function that returns an SDO_GEOMETRY object.

The use of the DIMINFO field or the tolerance changes based on which footprint is used. The first footprint requires that the diminfo array be passed into the function for the geometry object. This is the *required version* of the SDO_GEOM.SDO_ARC_DENSIFY function call *if the data has not been migrated to include four digit SDO_GTYPE values* which include the dimensionality.

SDO_GEOM.SDO_ARC_DENSIFY Function (continued)

If the geometry object has been specified with a four digit SDO_GTYPE value, then the tolerance footprint of the function can be used.

Both footprints require <params>. The <params> value is a quoted string containing a value for <arc-tolerance> and an optional parameter specifying the <unit> of the <arc-tolerance>.

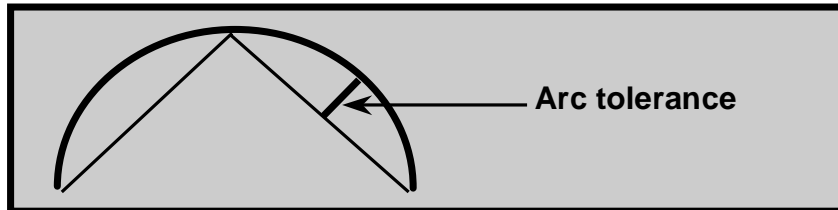
The <arc-tolerance> parameter is required. It specifies the maximum distance between the arc and the linestring generated to approximate it. If the <arc-tolerance> is specified as less than the tolerance, the tolerance value is used.

Both footprints optionally allow a parameter that specifies the <unit>. There are many different values allowed for the <unit> distance parameter, although some values have the same meaning (for example KM and KILOMETER). A list of supported values for the <unit> parameter is can be seen by doing the following query: select sdo_unit from mdsys.sdo_dist_units;

If both <arc-tolerance> and <unit> are specified, they should both be put into the same set of quotes, for example, 'unit=foot arc_tolerance=100'.

SDO_ARC_DENSIFY

ARC_TOLERANCE



- **ARC_TOLERANCE** is the maximum distance between the original arc, and the line string generated by Oracle to approximate the arc
- The **ARC_TOLERANCE** unit can be specified with the **UNIT** parameter
- If the **ARC_TOLERANCE** is smaller than **TOLERANCE**, an Oracle error is returned

ORACLE

9-21

Copyright © Oracle Corporation, 2001. All rights reserved.

ARC_TOLERANCE

ARC_TOLERANCE is the maximum allowable distance between the original arc, and the straight line segments generated by the function to approximate the arc.

ARC_TOLERANCE can be associated with a UNIT such as FOOT, METER, MILE, KILOMETER, and so on by specifying the UNIT parameter.

The lower bound for ARC_TOLERANCE is the tolerance value specified for the data.

SDO_ARC_DENSIFY: Example

- Densify a given arc
- Note the output geometry has no arcs

```
select sdo_geom.sdo_arc_densify (
    mdsys.sdo_geometry(2002, 32775, NULL,
        mdsys.sdo_elem_info_array(1, 2, 2),
        mdsys.sdo_ordinate_array(1832493.9, 2175555.32,
                                1832923.9, 2175985.32,
                                1833353.9, 2175555.32)),
    0.005, 'arc_tolerance=0.01 unit=mile')
densified_arc
from dual;

SDO_GEOMETRY(2002, 32775, NULL,
    SDO_ELEM_INFO_ARRAY(1, 2, 1),
    SDO_ORDINATE_ARRAY(1832493.90, 2175555.32,
                        1832551.51, 2175770.32,
                        1832708.90, 2175927.71,
                        1832923.90, 2175985.32,
                        1833138.90, 2175927.71,
                        1833296.29, 2175770.32,
                        1833353.90, 2175555.32))
```

ORACLE

9-22

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.SDO_ARC_DENSIFY Example

In this example, SDO_ARC_DENSIFY is called with a line string composed of circular arcs (note the 2,2 specified for the element type and the interpretation).

The ARC_TOLERANCE for this function is set to 0.01 miles, which is approximately 53 feet. The set of line segments generated to approximate the original circular arc geometry will never be more than 53 feet (about 16 meters) from the original circular arc.

The geometry that is generated as a result of this query changes the element type and interpretation to 2,1, which is line string composed of straight lines. Note that if the ARC_TOLERANCE was coarser (a larger value) then it is possible that fewer coordinates would be generated to approximate the circular arc. If the ARC_TOLERANCE was finer (a smaller number) then it is possible that more coordinates would be generated to approximate the circular arc.

Note that the tolerance is 0.005 meter, which is the coordinate system unit.

SDO_BUFFER Function

- **SDO_GEOM.SDO_BUFFER**
 - Generates a buffer polygon around a geometry
 - Takes an SDO_GEOMETRY object as input
 - Any kind (point, line, polygon, compound)
 - Can be the result of another function
 - Can buffer geodetic geometries
 - Returns an SDO_GEOMETRY object containing the buffer (polygon)

ORACLE

9-23

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_BUFFER

The SDO_GEOM.SDO_BUFFER function generates a buffer polygon around an input geometry.

The input geometry can be any type of SDO_GEOMETRY object supported by Oracle Spatial, including point, line, polygon, multipoint, multiline, multipolygon, compound line string, or compound polygon. Polygons with holes are supported.

The geometry object can come from a table, it can be a variable or a constructor, or it can be a geometry that is the result of another spatial function.

In geodetic space, the buffer operation will be approximate. The result is approximate rather than exact due to two factors:

1. The input geodetic geometry is implicitly projected to a local tangent plane, buffered, then projected back to the geodetic coordinate system.
2. Any arc geometries generated by the buffer operation are implicitly densified or smoothed to a number of line segments such that the distance between the original arc and the line segment never exceeds the ARC_TOLERANCE value specified by the user.

The result of the buffer operation is an SDO_GEOMETRY object that contains the buffered geometry. Depending on the geometry that was buffered, the resulting geometry will be one or more polygons.

SDO_BUFFER Function

```
SDO_GEOMETRY := SDO_GEOM.SDO_BUFFER
  ( <geometry>, <diminfo>,
    <distance> [, '<params>'])
or
SDO_GEOMETRY := SDO_GEOM.SDO_BUFFER
  ( <geometry>, <distance>,
    <tolerance> [, '<params>'])
```

- **<geometry> = SDO_GEOMETRY object to buffer**
 - Can be a variable or table column
 - Can be result of another function
- **<diminfo> = Dimension array**
- **<tolerance> = Number used as the tolerance**
- **<distance> = The buffer distance**

ORACLE

9-24

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_BUFFER Function

The SDO_GEOM.SDO_BUFFER function buffers a geometry by a specified distance, and returns the buffered geometry. SDO_GEOM.SDO_BUFFER is an overloaded function with two different footprints.

Both footprints require a parameter of type MDSYS.SDO_GEOMETRY. This parameter is the geometry to be buffered. It can come from a table, a variable or a constructor, or it can be the result of a function that returns an SDO_GEOMETRY object.

The use of the diminfo field or the tolerance changes based on which footprint is used. The first footprint requires that the diminfo array be passed into the function for the geometry object. This is the required version of the SDO_GEOM.SDO_BUFFER function call if the data has not been migrated to include four digit SDO_GTYPE values which include the dimensionality.

If the geometry object has been specified with a four digit SDO_GTYPE value, then the tolerance footprint of the function can be used. The tolerance is based on the precision of the data. With non-geodetic spatial data, the tolerance is specified in the units of the data. The tolerance for geodetic data is always specified in meters.

Both footprints require a distance value. The default unit for the distance parameter is the unit the data was defined in, or if the data is geodetic, the default unit is meter.

SDO_BUFFER Function

- **<params> =**
 - **<arc-tolerance>**
 - Only specify arc-tolerance if geometry to buffer is geodetic (required)
 - If arc-tolerance is smaller than tolerance, an Oracle error is returned
 - **<unit>** - Unit associated with distance and arc-tolerance. If UNIT is not specified:
 - For projected data, unit defaults to the coordinate system unit of the geometry being buffered
 - For geodetic data, unit defaults to meter
- **returns = an SDO_GEOMETRY object**

ORACLE

9-25

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_BUFFER Function (continued)

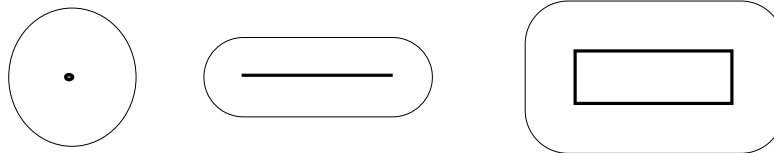
If the data is geodetic, the <arc-tolerance> parameter is required. It specifies the maximum distance between the arc generated during the buffer operation and the line string generated to approximate it. If the <arc-tolerance> value specified is less than the tolerance, an error is returned.

Both footprints optionally allow a parameter that specifies the <unit> for the results. This is formatted within a quoted string, for example, 'UNIT=FOOT' or 'UNIT=METER'. There are 54 different values allowed for the <unit> distance parameter, although some values have the same meaning (for example KM and KILOMETER). If the <unit> parameter is specified it is applied to the <distance> parameter. If an <arc-tolerance> is specified then the <unit> parameter is applied to <arc-tolerance> as well. A list of supported values for the <unit> parameter can be seen by doing the following query: select sdo_unit from mdsys.sdo_dist_units;

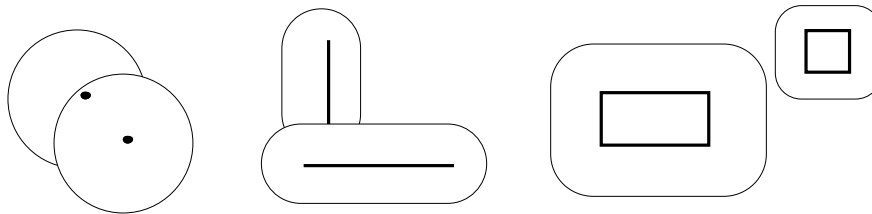
If both <arc-tolerance> and <unit> are specified, they should both be put into the same set of quotes, for example, 'unit=mile arc_tolerance=0.5'.

Buffer Examples

- **Simple geometries**



- **Collection geometries**



ORACLE

9-26

Copyright © Oracle Corporation, 2001. All rights reserved.

Buffer Examples

Shown are some examples of buffers generated around geometry objects.

Buffer Examples

Closed line



Polygon with void



ORACLE

9-27

Copyright © Oracle Corporation, 2001. All rights reserved.

Buffer Examples (continued)

Shown are some buffers around slightly more complicated geometry objects.

SDO_BUFFER Examples

- Generate a 25 kilometer buffer around highway I170

```
SELECT
  sdo_geom.sdo_buffer (geom, 25, 0.5,
                      'arc_tolerance=0.05 unit=km')
    buffer_geom
FROM geod_interstates
WHERE highway = 'I170';
```

- What is the area of the buffer in square kilometers?

```
SELECT
  sdo_geom.sdo_area (
    sdo_geom.sdo_buffer (geom, 25, 0.5,
                        'arc_tolerance=0.05 unit=km'),
    0.5, 'unit=sq_km') area_of_buffer
FROM geod_interstates
WHERE highway = 'I170';
```

ORACLE

9-28

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_BUFFER Examples

Shown are two queries that create a buffer around US Interstate I170. The first query creates a 25 kilometer buffer around Interstate I170 which is stored in a geodetic coordinate system. The arc tolerance for the query is 0.05 kilometers. This means the line segments that are implicitly generated by the buffer operation to approximate any circular arcs will be no further than 50 meters from the circular arc.

The second query wraps an SDO_GEOM.SDO_AREA function call around the same SDO_BUFFER previously completed. In this case, the area (in square kilometers) is returned for the 25 kilometer buffer created around Interstate I170.

SDO_BUFFER Examples

- Find all counties that have any interaction with a 25 kilometer buffer around highway 'I170'

```
SELECT /*+ ordered */ c.county
FROM geod_interstates i,
     geod_counties c
WHERE i.highway = 'I170'
     AND sdo_relate (
         c.geom,
         sdo_geom.sdo_buffer (
             i.geom, 25, 0.5,
             'ARC_TOLERANCE=0.05 UNIT=KILOMETER'),
         'mask=ANYINTERACT querytype=WINDOW') = 'TRUE';
```

- This can also be done using SDO_WITHIN_DISTANCE

ORACLE

9-29

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_BUFFER Examples (continued)

In this example, buffer is used with the SDO_RELATE operator. The exact same buffer is generated that was used in the previous queries. In this case, the buffered geometry (Interstate I170 buffered by 25 kilometers with an arc tolerance of 50 meters) is used as the window geometry in an SDO_RELATE operation, where all of the counties from the GEOD_COUNTIES layer are returned that have any interaction with the buffered I170.

Note that this query is conceptually the same as a query using the SDO_WITHIN_DISTANCE operator, where geometries with any interaction with a query window geometry buffered by a distance are returned.

SDO_BUFFER Examples

- Find the counties *entirely inside* a 25 kilometer buffer around highway 'I170'

```
SELECT /*+ ordered */ c.county, c.popsqmi
FROM geod_interstates i,
     geod_counties c
WHERE highway = 'I170'
     AND sdo_relate (c.geom,
                    sdo_geom.sdo_buffer ( i.geom, 25, 0.5,
                                           'ARC_TOLERANCE=0.05 UNIT=KM' ),
                    'mask=INSIDE querytype=WINDOW') = 'TRUE';
```

- This *cannot* be done using SDO_WITHIN_DISTANCE

ORACLE

9-30

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_BUFFER Examples (continued)

In this example, as in the last example, buffer is used with the SDO_RELATE operator. The exact same buffer is generated that was used in the previous queries. In this case, the buffered geometry (Interstate I170 buffered by 25 kilometers with an arc tolerance of 50 meters) is used as the window geometry in an SDO_RELATE operation, where all of the counties in the GEOD_COUNTIES layer are returned that are completely inside the buffered I170.

Note that this query is now very different from the SDO_WITHIN_DISTANCE operator, where in this case only county geometries that are completely inside a window geometry are returned, where SDO_WITHIN_DISTANCE would return all county geometries that had any interaction with the window geometry (the buffered I170).

SDO_BUFFER Examples

- Hint 4: Don't call functions in operators as was shown in the previous two slides. The spatial index gets disabled. Use the following syntax instead:

```
SELECT c.county, c.poppsqmi
FROM (SELECT /*+ no_merge */
      sdo_geom.sdo_buffer (
        i.geom,
        25,
        0.5, 'ARC_TOLERANCE=0.05 UNIT=KM') buffer_geom
      FROM geod_interstates i
      WHERE highway = 'I170') x,
     geod_counties c
WHERE sdo_relate (
      c.geom,
      x.buffer_geom,
      'mask=INSIDE querytype=WINDOW') = 'TRUE';
```

- This hint may not be necessary in Oracle9i

ORACLE

9-31

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_BUFFER Examples (continued)

The previous two queries showed the function called from within the SDO_RELATE operator. In versions of Oracle before 9i, this disabled the spatial index, so spatial operators would run slow. Queries should not be written with functions embedded in them.

The syntax shown is a rewritten version of the previous query. It shows the use of an in-line view with the no_merge optimizer hint. The in-line view creates an alias x.buffer_geom, which is the Interstate I170 buffered to 25 kilometers. The alias x.buffer_geom is what is used as the window geometry in the SDO_RELATE operator.

The use of the in-line view also requires the NO_MERGE optimizer hint. This hint is used to tell the Oracle optimizer not to move the buffer function back into the operator (do not merge the in-line view back into the where clause), thus disabling the spatial index.

Recap of Four Important Hints

Four hints to help ensure optimal performance:

- Always use = 'TRUE', never <> 'FALSE' or = 'true'.
- When using multiple masks, in rare cases when comparing geometries with lots of vertices, better performance might be achieved with multiple SDO_RELATEs, each with one mask, separated by ORs.
- Use the /*+ ORDERED */ hint when the query window comes from a table (not always necessary, but will never hurt performance).
- Use an in-line view and the /*+ NO_MERGE */ hint to ensure that functions are not used within operators (may not be necessary in Oracle9i).

ORACLE

9-32

Copyright © Oracle Corporation, 2001. All rights reserved.

A Recap of the Four Important Hints

1. Always use = 'TRUE' when testing spatial operators. Never use <> 'FALSE' or = 'true'.
2. When using multiple masks, in rare cases when comparing geometries with lots of vertices, better performance might be achieved with multiple SDO_RELATEs, each with one mask, separated by ORs rather than using the "+" syntax in a single SDO_RELATE operation. This is because interior tile optimizations are used for all single mask queries, where the only multiple masks queries that use interior tile operations are 'mask=inside+coveredby' and 'mask=contains+covers'.
3. Use the /*+ ordered */ hint when the query window comes from a table. This may not always be necessary, but it will never hurt performance.
4. Use an in-line view and the /*+ no_merge */ hint to make sure functions are not used within operators.

Following hints 1, 3, and 4 will never hurt performance, and will always ensure good performance.

Hint 2 requires testing using specific data sets to determine whether using the optimization results in better or worse performance.

Spatial Boolean Functions

ORACLE

9-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Boolean Functions

SDO_GEOM.SDO_UNION

- **Generates a geometry representing the union of two geometries.**



ORACLE

9-34

Copyright © Oracle Corporation, 2001. All rights reserved.

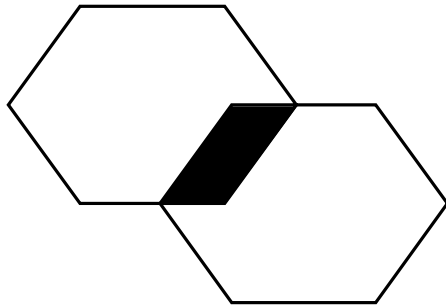
SDO_GEOM.SDO_UNION

This function takes two geometries as input, and returns the topological union of the two geometries. Common borders are removed. In the case shown, the two geometries that are UNIONed become a single polygon. If the two geometries were disjoint, a multipolygon would be returned.

Spatial Boolean Functions

SDO_GEOM.SDO_INTERSECTION

- **Generates a geometry representing the intersection of two geometries.**



ORACLE

9-35

Copyright © Oracle Corporation, 2001. All rights reserved.

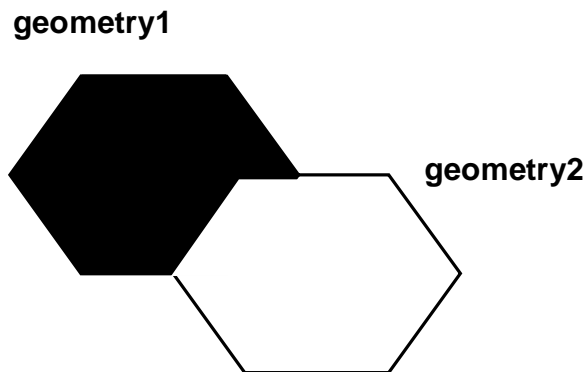
SDO_GEOM.SDO_INTERSECTION

The SDO_GEOM.SDO_INTERSECTION function returns points, lines, and areas in common between two geometries.

Spatial Boolean Functions

SDO_GEOM.SDO_DIFFERENCE

- Generates a geometry representing the difference between two geometries (geometry1 - geometry2).



ORACLE

9-36

Copyright © Oracle Corporation, 2001. All rights reserved.

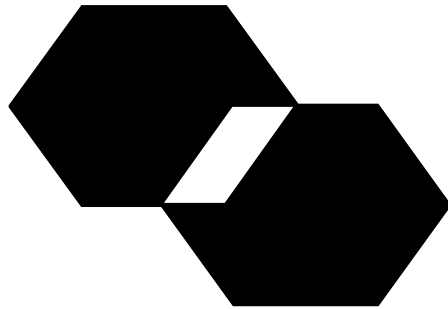
SDO_GEOM.SDO_DIFFERENCE

The SDO_GEOM.SDO_DIFFERENCE function subtracts the second geometry from the first geometry. If the first geometry has area (polygon or multipolygon), only subtracting another geometry with area will have an effect. If the first geometry is a line, only subtracting a polygon or a line will have an effect. If the first geometry is a point, then all geometry types can be subtracted from it.

Spatial Boolean Functions

SDO_GEOM.SDO_XOR

- **Generates a geometry representing the symmetric difference between two geometries.**



ORACLE

9-37

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_XOR

This function returns a geometry that consists of points, lines, and/or polygons where only one of the two geometries exist. Note that if one geometry is a polygon then the SDO_GEOM.SDO_XOR function will only have an effect if the other geometry is a polygon. In a similar fashion, lines will only be affected by other lines where the intersection is linear (not point), and points will only be affected by other points.

The SDO_<BOOLEAN> Functions

```
SDO_GEOMETRY := SDO_GEOM.SDO_<BOOLEAN>
    ( <geometry-1>, <diminfo-1>,
      <geometry-2>, <diminfo-2> )
or
SDO_GEOMETRY := SDO_GEOM.SDO_<BOOLEAN>
    ( <geometry-1>, <geometry-2>, <tolerance> )
```

- <BOOLEAN> = UNION, INTERSECTION, DIFFERENCE, or XOR
- <geometry-n> = SDO_GEOMETRY objects
 - Can be a variable or table column
- <diminfo-n> = dimension arrays
- <tolerance> = numeric tolerance for function
- returns = an SDO_GEOMETRY object

ORACLE

9-38

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_GEOM.SDO_<BOOLEAN> Functions

The BOOLEAN functions in Oracle Spatial apply boolean filters across two geometries, returning an SDO_GEOMETRY object the result of the boolean operation.

SDO_GEOM.SDO_<BOOLEAN> are overloaded functions with two different footprints.

Both footprints require parameters of type MDSYS.SDO_GEOMETRY. These parameter are the geometries to apply the boolean function to. In any combination they can come from a table, be variables or constructors, or they can be the result of functions that return an SDO_GEOMETRY object.

The use of the diminfo field or the tolerance changes based on which footprint is used. The first footprint requires that the diminfo arrays be passed into the function for the geometry objects. These are the required version of the SDO_GEOM.SDO_<BOOLEAN> function call if the data has not been migrated to include 4 digit SDO_GTYPE values which include the dimensionality.

If the geometry objects have been specified with a 4 digit SDO_GTYPE values, then the tolerance footprint of the functions can be used. The tolerance is based on the precision of the data. With non-geodetic spatial data, the tolerance is specified in the units of the data. The tolerance for geodetic data is always specified in meters.

The order in which the geometries are specified only matters for the SDO_GEOM.SDO_DIFFERENCE function, where <geometry-2> is subtracted from <geometry-1>. The order of the geometries doesn't matter for the other spatial boolean functions.

Putting it all Together

Calculate the area of the section of each county within the I170 buffer.

```
SELECT county,
       sdo_geom.sdo_area (
         sdo_geom.sdo_intersection (
           c.geom,
           sdo_geom.sdo_buffer (
             i.geom, 25, 0.5, 'unit=km arc_tolerance=0.05'),
           0.5, 'unit=sq_km') geom_area
FROM   geod_counties c,
       geod_interstates i
WHERE  i.highway = 'I170'
       AND sdo_within_distance(
         c.geom, i.geom, 'distance=25 unit=km') = 'TRUE';
```

ORACLE

9-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Putting it All Together

This example shows the calculation of the area of each county that is within the 25 kilometer buffer generated around Interstate I170. Starting with the WHERE clause, the SDO_WITHIN_DISTANCE operator is invoked to return all of the counties that have any interaction with Interstate I170. The counties returned by this portion of the query are passed to the top part of the query.

The innermost part of the top portion of the query is the same 25 kilometer buffer generated around Interstate I170 that has been worked with several times in this Lesson. Wrapping the SDO_GEOM.SDO_BUFFER is an SDO_GEOM.SDO_INTERSECTION which takes each of the counties returned from the SDO_WITHIN_DISTANCE and finds the intersection between the county and the buffered I170. At the highest level the select is done that gets the name of the county and area of the intersection between the county and the buffered Interstate I170.

The query would work without the SDO_WITHIN_DISTANCE in the where clause, but the intersection would be done for every county in the geod_counties layer, not just the counties that have an interaction with the buffered I170. In this case, the SDO_WITHIN_DISTANCE is used so that the spatial index is applied to quickly narrow down the the counties to only those counties that have some interaction with the I170 buffer.

Spatial Analysis Functions

- All return a geometry object
- **SDO_CENTROID**
 - “Center of gravity” function
 - Returns point geometry object
 - Not valid for multipolygons
- **SDO_POINTONSURFACE**
 - Returns the first point of the first element in SDO_ORDINATE_ARRAY
- **SDO_CONVEXHULL**
 - Returns convex polygon
 - Not valid for circles or geometries with less than three points (not on a straight line)

ORACLE

9-40

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Analysis Functions

There are three spatial analysis functions described in this section. All of them return a geometry object.

- SDO_GEOM.SDO_CENTROID is a “center of gravity” function that returns a point geometry object corresponding to the 2D center of gravity of a polygon. The input geometry for this function must be a single polygon.
- SDO_GEOM.SDO_POINTONSURFACE returns the first point of the first element in the SDO_ORDINATE_ARRAY.
- SDO_GEOM.SDO_CONVEXHULL returns a convex polygon around the input geometry. The input geometry must have at least three points, and all three points cannot be collinear. It can be thought of as tightly fitting an elastic band around the geometry object.

Spatial Analysis Functions

```
SDO_GEOMETRY := SDO_GEOM.SDO_<ANALYSIS>
                ( <geometry>, <diminfo> )
or
SDO_GEOMETRY := SDO_GEOM.SDO_<ANALYSIS>
                ( <geometry>, <tolerance> )
```

```
SELECT sdo_geom.sdo_centroid(c.geom, 0.5) centroid
FROM   geod_counties c
WHERE  c.county = 'Passaic' and c.state_abrv = 'NJ';
```

```
SELECT sdo_geom.sdo_pointonsurface(c.geom, 0.5) surface_point
FROM   geod_counties c
WHERE  c.county = 'Passaic' and c.state_abrv = 'NJ';
```

```
SELECT sdo_geom.sdo_convexhull(c.geom, 0.5) convexhull
FROM   geod_counties c
WHERE  c.county = 'Passaic' and c.state_abrv = 'NJ';
```

ORACLE

9-41

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Analysis Functions (continued)

All three functions have the same footprint. They accept a geometry object and either a numeric tolerance value or a diminfo array as the second parameter. If the data does not have a four-digit SDO_GTYPE where the dimensionality is encoded in the SDO_GTYPE then the diminfo footprint must be used.

The first query finds the centroid (or center of gravity) of Passaic County in New Jersey.

The second query returns the first point in the SDO_ORDINATE array of the Passaic County, New Jersey geometry object.

The third query returns a convex hull around Passaic County in New Jersey.

Spatial MBR Functions

- **SDO_MBR**
 - Returns the minimum bounding rectangle around a geometry object
 - Returns an optimized rectangle
- **SDO_MAX_MBR_ORDINATE**
 - Returns the maximum ordinate for the dimension specified
- **SDO_MIN_MBR_ORDINATE**
 - Returns the minimum ordinate for the dimension specified

MBR functions are not supported for geodetic data!

ORACLE

9-42

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial MBR Functions

Oracle Spatial includes three functions that return information about minimum bounding rectangles.

`SDO_GEOM.SDO_MBR` returns the minimum bounding rectangle around a geometry object. The minimum bounding rectangle is returned as an optimized rectangle geometry (element type 1003, interpretation 3).

`SDO_GEOM.SDO_MAX_MBR_ORDINATE` returns the maximum ordinate for the dimension specified. For instance, if the dimension specified is 1 then the function looks at the first ordinate of all the vertices in the geometry and returns the largest value for that ordinate.

`SDO_GEOM.SDO_MIN_MBR_ORDINATE` returns the minimum ordinate for the dimension specified. For instance, if the dimension specified is 2 then the function looks at the second ordinate of all the vertices in the geometry and returns the smallest value for that ordinate.

Spatial MBR functions are not supported for geodetic data (data with geodetic spatial reference system id values).

Spatial MBR Functions

```
SDO_GEOMETRY := SDO_GEOM.SDO_MBR  
    ( <geometry> [, <diminfo>] )
```

```
number := SDO_GEOM.SDO_MAX_MBR_ORDINATE  
    ( <geometry>, [ <diminfo> , ]  
    <dimension_number> )
```

```
number := SDO_GEOM.SDO_MIN_MBR_ORDINATE  
    ( <geometry>, [ <diminfo> , ]  
    <dimension_number> )
```

- **<geometry> = SDO_GEOMETRY object**
 - can be a variable or table column
- **<diminfo> = dimension array**
- **<dimension_number> = dimension to return the maximum or minimum ordinate for**

ORACLE

9-43

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial MBR Functions (continued)

SDO_GEOM.SDO_MBR - The first argument to this function is a geometry object containing the geometry to get the minimum bounding rectangle of. If the geometry does not have the dimensionality encoded in the SDO_GTYPE (using the four-digit SDO_GTYPE) then the diminfo array must be passed into the function as well as the second, optional argument. The function returns an SDO_GEOMETRY object with an optimized rectangle that is the minimum bounding rectangle around the input geometry.

The functions SDO_GEOM.SDO_MAX_MBR_ORDINATE and SDO_GEOM.SDO_MIN_MBR_ORDINATE both take the same parameters. Like SDO_GEOM.SDO_MBR they both have as the first argument a geometry object containing the geometry to get the maximum or minimum ordinate of. If the geometry does not have the dimensionality encoded in the SDO_GTYPE (using the four-digit SDO_GTYPE) then the diminfo array must be passed into the function as well as the second, optional argument. The final required argument is the dimension number. For instance, if the dimension number is 1, then the maximum or minimum value of the ordinate in the first dimension is returned. If the dimension number is 2, then the maximum or minimum value of the ordinate in the second dimension is returned.

Spatial MBR Function Examples

- **SDO_GEOM.SDO_MBR**

```
SELECT sdo_geom.sdo_mbr(c.geom) mbr
FROM proj_counties c
WHERE c.county = 'Passaic' and c.state_abrv = 'NJ';
```

- **SDO_GEOM.SDO_MIN_MBR_ORDINATE**

```
SELECT sdo_geom.sdo_min_mbr_ordinate (c.geom, 2) min_y
FROM proj_counties c
WHERE c.county = 'Passaic' and c.state_abrv = 'NJ';
```

ORACLE

9-44

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial MBR Function Examples

The SDO_GEOM.SDO_MBR example takes as input the geometry used for Passaic County in the US state of New Jersey selected from the projected counties table (note the geometry is not geodetic), and returns the minimum bounding rectangle geometry that encloses Passaic County.

The function SDO_GEOM.SDO_MIN_MBR_ORDINATE returns the minimum value stored in the second dimension (typically the y dimension) of the geometry representing Passaic County in New Jersey.

Spatial Aggregate Functions

- Operate on a set of geometries
- Aggregate geometry returned based on operation
- Simplifies coding
- Users can define their own aggregate functions using Oracle's user-defined aggregate framework
- Require the 4-digit SDO_GTYPE

ORACLE

9-45

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Aggregate Functions

Spatial aggregate functions operate on a set of geometries rather than just one or two geometries that other functions work with. They return a single geometry object that is an aggregation of the set of input geometries based on the aggregate operation specified.

Use of spatial aggregate functions simplifies coding and increases the speed of operations that used to require programming to accomplish.

Additional spatial aggregate functions can be created using Oracle's user-defined aggregate framework.

Spatial aggregate functions are the first functions within Oracle Spatial that require the four-digit SDO_GTYPE values that include dimensionality of the geometry. This means that data must be in the Oracle Spatial 8.1.6 and higher format to work with spatial aggregate functions.

SDOAGGRTYPE

- **New type in Spatial**
 - Input parameter to most spatial aggregate functions
 - Can reuse type for user-defined aggregates

```
CREATE TYPE SDOAGGRTYPE AS OBJECT (  
    geometry    MDSYS.SDO_GEOMETRY,  
    tolerance NUMBER);
```

- **Note:** This type gets created when you install Oracle Spatial. You do not have to create this type.

ORACLE

9-46

Copyright © Oracle Corporation, 2001. All rights reserved.

MDSYS.SDOAGGRTYPE

A new type is created when Oracle Spatial is installed in Oracle9i. This new type is MDSYS.SDOAGGRTYPE, and it consists of a geometry object and a numeric tolerance value. The MDSYS.SDOAGGRTYPE is the input parameter type for most spatial aggregate functions, and it can be reused if a user-defined spatial aggregate function is created.

SDO_AGGR_UNION

- **Returns a geometry object that is the topological union of the set of input geometries**
- **Aggregating geometries in the server can reduce data transferred to the client**
- **Eliminates the need for PL/SQL code to UNION a set of geometries**

ORACLE

9-47

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_UNION

The SDO_AGGR_UNION function takes as input a set of geometry objects and a tolerance value (in an object of type MDSYS.SDOAGGRTYPE) and performs a union operation across the complete set of geometry objects. The SDO_AGGR_UNION operation is a fast and easy way of UNIONing a set of geometries. In the past, aggregate union operations either had to be done in client software, which required potentially large amounts of data to be moved across the network, or required custom code be written.

SDO_AGGR_UNION Example

- Union all the county boundaries for the states of New York and New Jersey, and generate a geometry for each state

```
SELECT sdo_aggr_union(mdsys.sdoaggrtype(a.geom, 0.5)),  
       a.state  
FROM   geod_counties a  
WHERE  a.state_abrv in ('NY', 'NJ')  
  
GROUP by state;
```

ORACLE

9-48

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_UNION Example

This example unions all of the counties in New York and New Jersey, and returns a single aggregated geometry and the state name for each of the two states. Note the input type to the SDO_AGGR_UNION function is of type MDSYS.SDOAGGRTYPE, which consists of the geometry and the numeric tolerance for the geodetic layer.

SDO_AGGR_UNION Example

- Union all the county boundaries which intersect Highway 'I93', and generate a polygon for each state's counties

```
SELECT /*+ ordered */
      sdo_aggr_union(mdsys.sdoaggrtype(a.geom, 0.5)) aggr_geom,
      a.state
FROM   geod_interstates b,
      geod_counties a
WHERE  b.highway = 'I93'
      AND sdo_relate(a.geom, b.geom,
                     'mask=ANYINTERACT querytype=WINDOW')='TRUE'
GROUP by a.state;
```

ORACLE

9-49

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_UNION Example (continued)

In this example all of the counties that intersect highway I93 are selected, and for each state a polygon is generated that is the union of the counties that intersect highway I93.

In the WHERE clause there is an SDO_RELATE operator, which uses the spatial index to determine which counties have any interaction with highway I93.

The counties that intersect Interstate I93 are then UNIONed together on a state-by-state basis, using the SDO_AGGR_UNION function to return one set of aggregated counties per state.

SDO_AGGR_CENTROID

- Returns a point that is the “center of gravity” of the set of input geometries
- Centroid can be computed for
 - Set of points and multipoints
 - Set of polygons and multipolygons
- Example: Can be used to find the label location for a set of map features

ORACLE

9-50

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_CENTROID

The SDO_AGGR_CENTROID function takes as input a set of points, multipoints, polygons, and multipolygons. It returns the weighted center of gravity in two dimensions of all of the input features.

All points are given equal weight when determining the center of gravity. Polygons are weighted by area. If a layer has both points and polygons, since points have no area they are not used when determining the centroid.

An example of using the SDO_AGGR_CENTROID aggregate function is to find a label location for a set of features when building a map.

SDO_AGGR_CENTROID Example

- For each state, find the centroid of the cities in that state

```
SELECT a.state_abrv,  
       sdo_aggr_centroid(mdsys.sdoaggrtype(a.location,0.5))  
FROM geod_cities a  
GROUP by state_abrv;
```

ORACLE

9-51

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_CENTROID Example

This example finds the centroid of all of the cities in a state, returning the state abbreviation and a geometry object for each state in the geod_cities layer. As in the SDO_AGGR_UNION aggregate function, the input parameter for SDO_AGGR_CENTROID is of type MDSYS.SDOAGGRTYPE.

To simply get an aggregated centroid of all of the cities in the geod_cities layer the query can be rewritten as follows:

```
select sdo_aggr_centroid(mdsys.sdoaggrtype(a.location, 0.5))  
from geod_cities a;
```

Note the centroid is an MDSYS.SDO_GEOMETRY point object.

SDO_AGGR_CENTROID Example

- Find the centroid for the New England states

```
SELECT
    sdo_aggr_centroid( mdsys.sdoaggrtype(a.geom, 0.5)),
    'New England'
FROM geod_states a
WHERE a.state_abrv in ('MA','NH','VT','RI','CT','ME');
```

- This can be used to find the label location for the U.S. New England states

ORACLE

9-52

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_CENTROID Example (continued)

This example finds the centroid of the US New England States which can then be used as a label location for a display.

The where clause selects the geometries of the New England States from the geod_states layer, and the centroid of that set of states is calculated using the SDO_AGGR_CENTROID aggregate function. A single point is returned from the function, which is the centroid of the New England States.

SDO_AGGR_CONVEXHULL

- **Returns the convex hull around the set of input geometries**
- **Convex hull can be computed for**
 - Lines
 - Polygons
- **Example: Can be used to generalize a set of geometries**

ORACLE

9-53

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_CONVEXHULL

The SDO_AGGR_CONVEXHULL function takes as input a set of polygons and it returns the convex hull around all of the input features. The returned MDSYS.SDO_GEOMETRY object is a polygon.

An example of using the SDO_AGGR_CONVEXHULL aggregate function is to generalize a set of features where the accuracy of SDO_AGGR_UNION is not required.

A convex hull will never have any concave surfaces.

SDO_AGGR_CONVEXHULL Example

- Returns a geometry that is the convex hull of the New England states

```
SELECT
    sdo_aggr_convexhull( mdsys.sdoaggrtype(a.geom, 0.5))
FROM geod_states a
WHERE a.state_abrv in('MA','NH','VT','ME','RI','CT');
```

ORACLE

9-54

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_CONVEXHULL Example

In this example a convex hull is returned that encompasses the entire set of New England states. The convex hull is an approximation of the set of states.

SDO_AGGR_MBR

- Returns the minimum bounding rectangle of the set of input geometries
- This will work with any type of geometry
- Useful for finding the extents of a set of geometries
- *Not supported for geodetic data!*

ORACLE

9-55

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_MBR

The SDO_AGGR_MBR aggregate function returns an SDO_GEOMETRY object containing an optimized rectangle which is the minimum bounding rectangle around the set of input geometries.

The SDO_AGGR_MBR function works with all types of geometries, including points, lines, and polygons. This function is useful for finding the extent around a set of geometries.

Note: This function is not supported with geodetic data.

SDO_AGGR_MBR Example

- For a set of states, generate an MBR for each state's counties

```
SELECT sdo_aggr_mbr(a.geom), a.state
FROM proj_counties a
WHERE a.state_abrv in ('MA', 'NH', 'VT', 'ME')
GROUP by state;
```

Note: Input to this function is the SDO_GEOMETRY type,
not the SDOAGGRTYPE type.

ORACLE

9-56

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_AGGR_MBR Example

In this example, a minimum bounding rectangle is generated around all of the counties for each state in the where clause. There is one rectangle returned for each state.

Note that this spatial aggregate function has an input type of MDSYS.SDO_GEOMETRY, not the MDSYS.SDOAGGRTYPE of all of the previously described spatial aggregate functions.

Summary

In this lesson, you should have learned how to:

- **Use the advanced spatial functions to perform complex analyses**
- **Perform length and area calculations**
- **Perform distance calculations**
- **Generate buffers around geometries**
- **Perform geometry manipulations**
- **Use spatial aggregate functions**

ORACLE

10

Advanced Coordinate Systems Concepts

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Understand the benefits of the Whole Earth geometry model**
- **Understand and implement coordinate systems transformations**
- **Understand the process for creating user-defined coordinate systems**
- **Understand local coordinate systems**

ORACLE

Whole Earth Geometry Model

Accurate operations using geodetic data

- Before Oracle9i Spatial, you needed to project geodetic data for accurate length, distance, and area calculations
- In Oracle9i Spatial, the whole Earth model provides accurate length, distance and area calculations on geodetic data
- A large number of *length* and *area* units are supported for geodetic data calculations

ORACLE

10-3

Copyright © Oracle Corporation, 2001. All rights reserved.

The Whole Earth Geometry Model

Oracle9i Spatial returns accurate lengths, areas, and distances for both projected and geodetic data. In previous versions of Oracle Spatial (before Oracle9i), length, area, and distance calculations were only accurate for nongeodetic data. Before Oracle9i, to get accurate lengths, areas, and distances, geodetic data would have to be projected to a planar surface. Oracle9i Spatial's whole Earth geometry model takes into account the curvature of the Earth's surface when performing length, area, and distance calculations on geodetic data.

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

Whole Earth Geometry Model

- Supports geometries that span the 180 degree meridian and poles
- **BUFFER, CONVEXHULL, and CENTROID** are all supported via approximations using local projections
 - Buffers generated are densified (that is, contain no arcs or circles, only geometries consisting of straight lines)
- The Whole Earth geometry model requires coordinate system bounds to be (-180 to 180), (-90 to 90)

ORACLE

10-4

Copyright © Oracle Corporation, 2001. All rights reserved.

The Whole Earth Geometry Model (continued)

The Whole Earth geometry model can store, index, and operate on geodetic geometries, even if they span the 180 degree meridian and the poles. Additionally, SDO_GEOM.SDO_BUFFER, SDO_GEOM.SDO_CONVEXHULL, and SDO_GEOM.SDO_CENTROID are all supported using geodetic data via approximations that use projections to a local tangent plane. Also, SDO_GEOM.SDO_BUFFER automatically densifies the generated buffered geometries for geodetic data so that no arcs or circles are included in the final geometry.

The Whole Earth geometry model requires the coordinate system bounds be set to (-180 to 180) for longitude, and (-90 to 90) degrees for latitude.

Tolerance Revisited: Projected versus Geodetic

- Tolerance always has an associated distance UNIT
- In projected space, the tolerance has the same unit as the data's coordinate system
- In geodetic space, the data unit and tolerance unit are different
 - Data is in angular units (longitude/latitude)
 - Tolerance *must* always be in METERS (for example, 0.5 = 1/2 meter resolution)

ORACLE

10-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Tolerance Revisited: Projected versus Geodetic

Before Oracle9i Spatial, all tolerance values were specified in the same unit as the data, regardless of what the unit was for the input data.

In Oracle9i, for projected data, and for data which has no SRID value set (no associated coordinate system) the tolerance is still specified in the same coordinate system as the data.

In Oracle9i, for geodetic data, the data unit and the tolerance unit are not the same. In geodetic space, the data is in angular units of longitude and latitude (sometimes referred to as decimal degree), but the tolerance value is always in meters.

Why is a geodetic coordinate system special when it comes to tolerance? Because the distance associated with degrees of longitude varies as the location changes north or south. For example, starting at the equator, as you move a degree of latitude towards a pole, the distance it represents gets smaller and smaller until it is 0 kilometers at the pole. A degree of longitude at the equator is approximately 111 kilometers.

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

Migration from 8.1.x: Some Considerations

- **Semantics change**
 - Tolerance unit for geodetic layers
- **Optimized rectangles are not supported**
- **Results may vary**
 - Distances will be correct in Oracle9i Spatial for both geodetic and projected layers
 - In 8.1.x, only projected layers generated accurate distances
- **Indexes should change**
 - Reindex geodetic layers with R-trees

ORACLE

10-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Migration from Oracle Spatial 8.1.x to Oracle9i: Some Considerations

There are several things that must be taken into account when migrating geodetic layers from Oracle Spatial 8.1.x to Oracle9i Spatial:

- The semantics for tolerance have changed, and must be taken into account. As mentioned previously, all tolerance values for geodetic data must now be specified in meters. Not accounting for this change will likely result in slower indexing and query times. A tolerance of 0.0000005 for a geodetic layer now means a tolerance of 500 nanometers.
- Optimized rectangles are not supported
- Results may vary
 - Distances, lengths, and areas will be accurate in Oracle9i Spatial for both geodetic and projected layers
 - In Oracle 8.1.x, only projected layers generated accurate distances
- Reindex geodetic layers with R-tree indexes

Coordinate System Transformations

ORACLE

10-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Coordinate System Transformation Functions

Three functions supplied:

- **SDO_CS.TRANSFORM**
 - Transforms a geometry from one coordinate system to another
- **SDO_CS.TRANSFORM_LAYER**
 - Transforms a layer from one coordinate system to another
- **SDO_CS.VIEWPORT_TRANSFORM**
 - Convenience routine for visualizer applications. Transforms a viewport MBR to work with geodetic coordinate systems

ORACLE

10-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Coordinate System Transformation Functions

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

Oracle Spatial supplies three coordinate system transformation functions:

SDO_CS.TRANSFORM works with a single geometry at a time, and transforms that geometry from one coordinate system to another.

SDO_CS.TRANSFORM_LAYER takes all of the geometries in a layer and transforms the data from one coordinate system to another.

SDO_CS.VIEWPORT_TRANSFORM is a convenience routine for visualizer applications. This function accepts a viewport rectangle as input and transforms the rectangle to a valid polygon in a given coordinate system.

The SDO_CS.TRANSFORM Function

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

- **<geom> = geometry of type SDO_GEOMETRY**
 - Can be a variable or table column
 - Can be the result of another function
- **<diminfo> = dimension array**
- **<to_srid> = spatial reference system id to transform to**
- **return value = a geometry**

ORACLE

10-9

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_CS.TRANSFORM Function

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

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

If the geometry does not have a 4-digit SDO_GTYPE that includes the dimensionality of the geometry, then the diminfo array must be passed in as the second argument.

The SRID to transform the input geometry to is specified as the <to_srid>.

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

SDO_CS.TRANSFORM Example

- **Single geometry transformed, for example, Hillsborough County in New Hampshire**

```
select sdo_cs.transform (geom, 82151)
from proj_counties
where county = 'Hillsborough' and
      state = 'New Hampshire';
```

Note: All transformations require valid SDO_SRID field set in source geometry

Note: 82151 = "New Hampshire 2800 (1983, meters)" - State Plane CS 1983

ORACLE

10-10

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.TRANSFORM Example

In this example Hillsborough County in New Hampshire is transformed from an "Equal-Area Projection (United States)" coordinate system to "New Hampshire 2800 (1983, meters)" -- US State Plane Coordinate System 1983.

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

The SDO_CS.TRANSFORM_LAYER Procedure

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

- <table_name> = table with column of type SDO_GEOMETRY
- <geom_column_name> = column of type SDO_GEOMETRY
- <destination_table> = the table to be created by the procedure
- <to_srid> = spatial reference system id to transform to

ORACLE

10-11

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_CS.TRANSFORM_LAYER Procedure

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

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

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

The <to_srid> is the spatial reference system id to transform the layer to.

Note: This procedure requires that the input layer have a valid SDO_SRID value set, both in the individual geometries and at the layer level in the SRID field of USER_SDO_GEOM_METADATA.

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

TRANSFORM_LAYER Example

Entire layer transformed

- **<destination_table>** created by the utility
- **<destination_table>** will contain two columns:
 - GEOMETRY
 - SDO_ROWID

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

ORACLE

10-12

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.TRANSFORM_LAYER Example

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

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

Transformations

- **With operators (implicit transformations)**
 - **Second argument automatically transformed to same SDO_SRID of first argument if SDO_SRID in geometries are different**
- **With functions (explicit transformations)**
 - **Must manually transform geometries to same SRID (if 2 geometries)**

Note: For operators and functions to work correctly, one of the following must be true:

- **Both geometries must have georeferenced coordinate systems**
- **Both geometries must have local (non-Earth) coordinate systems**
- **Both geometries have NULL coordinate system information (NULL SRID)**

ORACLE

10-13

Copyright © Oracle Corporation, 2001. All rights reserved.

Transformations With Operators and Functions

Oracle Spatial has the following spatial operators:

- SDO_FILTER
- SDO_RELATE
- SDO_NN
- SDO_WITHIN_DISTANCE

Think of Oracle Spatial operators as having a template. The first argument of a spatial operator is always the column being searched and the second argument to a spatial operator is the window, or area of interest.

When an operator is run, if the SDO_SRID of the window geometry does not match the SDO_SRID of the layer being searched, Oracle Spatial will implicitly transform the window geometry to the same coordinate system (based on SDO_SRID value) as the layer being searched.

When functions require two geometries as input, the application or function caller must ensure that both geometries are in the same coordinate system. If the coordinate systems of the two geometries do not match, then the application must explicitly call the SDO_CS.TRANSFORM function.

Explicit or implicit transformations require that either:

- Both geometries have georeferenced coordinate systems, or
- Both geometries have local (non-Earth) coordinate systems

If both geometries have NULL SDO_SRID values, the results will be based on Cartesian calculations.

Transformation Example

- Union two geometries, each coming from tables with different SRIDs
 - Hillsborough County from the geod_counties table
 - Sullivan County from the proj_counties table
- When calling functions (for example, SDO_UNION) both geometries must have the same SRID

```
SELECT sdo_geom.sdo_union (
          sdo_cs.transform ( a.geom,
                             32775 ),
          b.geom, 0.0005)
FROM geod_counties a,
     proj_counties b
WHERE a.state_abrv = 'NH'
     AND a.county = 'Hillsborough'
     AND b.state_abrv = 'NH'
     AND b.county = 'Sullivan';
```

ORACLE

Transformation Example

In this example, two geometries are being UNIONed. Both have georeferenced coordinate systems, with the coordinate system of Hillsborough County in New Hampshire specified as geodetic ("Longitude / Latitude (WGS 84)"), and the coordinate system of Sullivan County in New Hampshire specified as projected ("Equal-Area Projection (United States)"). In this example, the geodetic geometry is transformed to the same coordinate system as the projected data, and then the union function (SDO_UNION) is applied. The geometries passed into the SDO_UNION function must be in the same coordinate system.

The SDO_CS.VIEWPORT_TRANSFORM Function

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

Note: SDO_CS.VIEWPORT_TRANSFORM is a convenience function used with any optimized rectangle. It returns a valid polygon for any given coordinate system.

ORACLE

10-15

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_CS.VIEWPORT_TRANSFORM Function

As discussed in the “Coordinate Systems Overview” lesson, geodetic layers in Oracle Spatial have some restrictions:

- Optimized rectangles (specified using only the lower left and upper right coordinates of the rectangle) are not allowed
- 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 in a regular Cartesian coordinate system (with the 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.

The 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**
- **returned value =**
 - If <to_srid> is a geodetic SRID, returns a geometry (not an optimized rectangle) that conforms to the rules of a geodetic geometry (for example, 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

10-16

Copyright © Oracle Corporation, 2001. All rights reserved.

The SDO_CS.VIEWPORT_TRANSFORM Function

This function is very useful for visualizer applications. Visualizer applications often display geodetic data in a regular Cartesian coordinate system (with the 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. Oracle Spatial provides the SDO_CS.VIEWPORT_TRANSFORM function as a convenience routine for visualizer applications due to the following restrictions:

- Optimized rectangles (specified using only the lower left and upper right coordinates of the rectangle) are not allowed
- Polygons must always be less than half the surface area of the Earth

SDO_CS.VIEWPORT_TRANSFORM

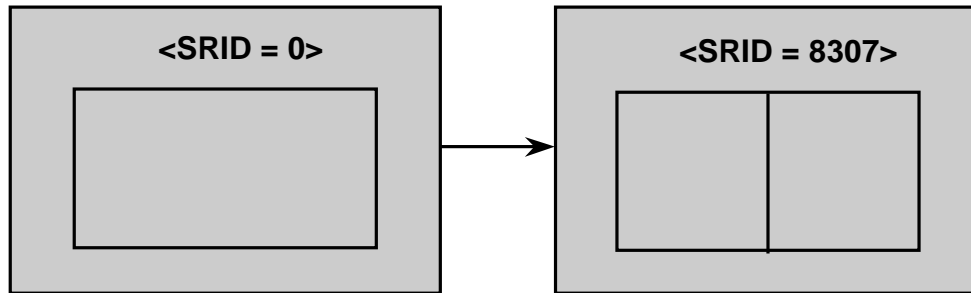
<geom> must be an optimized rectangle with SRID set to 0.

<to_srid> This is the SRID to transform the optimized rectangle to.

<returns>

- If the <to_srid> is a geodetic SRID, the function returns a geometry that is not an optimized rectangle. The geometry returned will conform to the rules of a geodetic geometry (it will be 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, the function returns the same optimized rectangle with the SRID set to <to_srid>.

SDO_CS.VIEWPORT_TRANSFORM When <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

10-17

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.VIEWPORT_TRANSFORM When <to_srid> is Geodetic

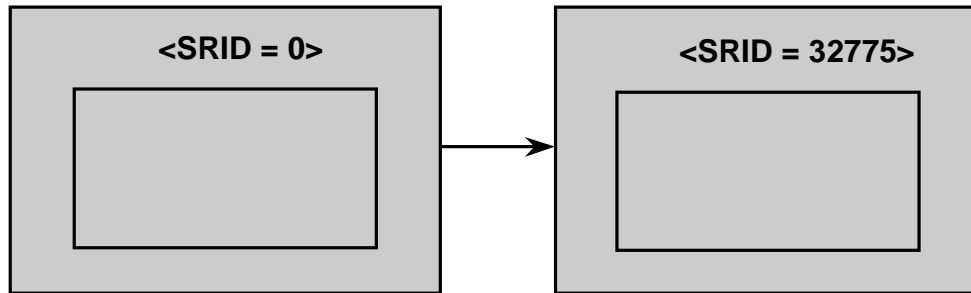
This is an example of passing in an optimized rectangle to SDO_CS.VIEWPORT_TRANSFORM, when the <to_srid> is geodetic.

Note: 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.

SDO_CS.VIEWPORT_TRANSFORM When <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

10-18

Copyright © Oracle Corporation, 2001. All rights reserved.

SDO_CS.VIEWPORT_TRANSFORM When <to_srid> is Projected

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 it's SRID set to <to_srid>.

The new geometry can be used in SDO_FILTER or SDO_RELATE with any valid mask.

Explicit Viewport Transformation

- **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 source viewport geometry must be an optimized rectangle.

ORACLE

10-19

Copyright © Oracle Corporation, 2001. All rights reserved.

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.

Indexing Geodetic Data

ORACLE

10-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Indexing Geodetic Data

- **A geodetic index:**
 - Supports geometries that cross the 180 degree meridian
 - Preserves distances and spatial relationships of geometries on the surface of the Earth
- **Only R-tree indexes can be geodetic**
- **By default, a geodetic R-tree index is created on geodetic data**
- **By default, quadtree indexing is disabled for geodetic data**
 - Can be overridden, but not recommended

ORACLE

10-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Indexing Geodetic Data

A geodetic spatial index is different from a nongeodetic spatial index in some important ways:

- A geodetic spatial index can index geometries that span the 180 degree meridian
- A geodetic index must preserve distance and all spatial relationships between geometries on the surface of the Earth

In Oracle Spatial, only R-tree indexes can be geodetic indexes.

If spatial data has a geodetic SRID associated with it, then by default a geodetic R-tree index is created.

By default, quadtree indexing is disabled for geodetic data. This can be overridden for backward compatibility with Oracle8i by specifying the “GEODETIC = FALSE” parameter in the CREATE INDEX statement. Setting “GEODETIC = FALSE” is not recommended, and is discussed in more detail in the next slide.

If an attempt is made to build a quadtree index using geodetic data without specifying “GEODETIC = FALSE”, an Oracle error is returned.

Indexing Geodetic Data

A new keyword, GEODETIC, is introduced in the CREATE INDEX syntax

- **By setting GEODETIC=FALSE, you can create a nongeodetic index on geodetic data**
- **This is the only way to create a quadtree index on geodetic data**
- **A nongeodetic index on geodetic data:**
 - **Can produce inaccurate results with spatial operators**
 - **Disables the UNIT parameter for SDO_WITHIN_DISTANCE and SDO_NN_DISTANCE**
- **Use of the GEODETIC keyword is not recommended**

ORACLE

10-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Indexing Geodetic Data

The GEODETIC keyword was introduced in the CREATE INDEX syntax of Oracle9i to allow a nongeodetic index to be built on geodetic data for backward compatibility.

When GEODETIC=FALSE is specified, then a nongeodetic spatial index is created on geodetic data.

Specifying GEODETIC=FALSE is the only way to create a quadtree index on geodetic data. If GEODETIC=FALSE is specified for an R-tree index, a nongeodetic R-tree index will be created.

If a nongeodetic index is created on geodetic data then:

- Inaccurate results can be produced when using spatial operators
- The UNIT parameter cannot be used with SDO_WITHIN_DISTANCE and SDO_NN_DISTANCE

Because of the possibility of returning inaccurate results, Oracle does not recommend setting GEODETIC=FALSE.

UNIT Support in Oracle Spatial

ORACLE

10-23

Copyright © Oracle Corporation, 2001. All rights reserved.

UNIT Support in Oracle Spatial

- **UNITs are supported for**
 - SDO_LENGTH
 - SDO_AREA
 - SDO_DISTANCE
 - SDO_BUFFER
 - SDO_WITHIN_DISTANCE
 - SDO_NN_DISTANCE
- **All operations have default units**
 - For projected data: The default is the unit associated with the projected coordinate system
 - For geodetic data: The default is meters
 - For local coordinate system: The default is the unit specified in the local coordinate system

ORACLE

10-24

Copyright © Oracle Corporation, 2001. All rights reserved.

UNIT Support in Oracle Spatial

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

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

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

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

UNIT Support in Oracle Spatial

Supported UNITS are in the following tables:

- **MDSYS.SDO_DIST_UNITS**
- **MDSYS.SDO_AREA_UNITS**

ORACLE

10-25

Copyright © Oracle Corporation, 2001. All rights reserved.

UNIT Support in Oracle Spatial

Distance units include:

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

Area units include:

SQ_METER, SQ_KILOMETER, SQ_CENTIMETER, SQ_MILLIMETER, SQ_CHAIN, SQ_FOOT,

SQ_INCH, SQ_LINK, SQ_MILE, SQ_ROD, SQ_SURVEY_FOOT, SQ_YARD, ACRE, HECTARE,

PERCH, and ROOD.

The SDO_UNIT column in the table MDSYS.SDO_DIST_UNITS has a list of the length and distance units supported.

The SDO_UNIT column in the table MDSYS.SDO_AREA_UNITS has a list of area units supported.

Accuracy of Geodetic Area Calculations

Area calculation examples

- **Areas the size of New Hampshire: within 0.0001%**
 - Area of New Hampshire is 24043 +/- 0.024 km²
- **Areas the size of India (approx. 1/3 size of US): within 0.001%**
 - Area of India is approximately 3277093 +/- 32.77 km²
- **Half the area of the Earth (biggest polygon): within 0.1%**
 - Approximately 255,082,311 +/- 255,082 km²
- **Larger differences in latitude introduce larger error margins, not to exceed the values listed**

ORACLE

10-26

Copyright © Oracle Corporation, 2001. All rights reserved.

Accuracy of Geodetic Area Calculations

The accuracy of the geodetic functions supplied with Oracle Spatial vary.

Area

- The accuracy of area calculations depends on geometry size and the how much it varies in Latitude. Larger differences in latitude introduce larger error margins, not to exceed the error margins listed.
- Smaller sized areas (for example, New Hampshire) are accurate to within 0.0001% (about 1 part in a million). For New Hampshire the area is approximately 24,042.6439 square kilometers +/- 0.024 square kilometers.
- Larger sized areas (for example, India), are accurate to within 0.001% (about 1 part in 100,000). For India the area is approximately 3277093.11 square kilometers +/- 32.77 square kilometers.
- An area covering half the Earth's surface is accurate to within 0.1%. For half the Earth's surface the area is approximately 255,082,311 square kilometers +/- 255,082 square kilometers.

Accuracy of Geodetic Length and Distance Calculations

- **Length calculations**
 - Generalized algorithm for short or long lengths
 - Accurate to within 0.00000001%
- **Distance calculations**
 - Point-to-point distances are same as Length, above
 - If not point-to-point, the algorithm is accurate to within 0.2%, and is generalized for short or long distances

ORACLE

10-27

Copyright © Oracle Corporation, 2001. All rights reserved.

Accuracy of Geodetic Length and Distance Calculations

The following describes the accuracy of the geodetic length and distance functions supplied with Oracle Spatial:

- **Length**
 - The algorithm for determining length values has been generalized for short or long lengths, and is accurate to 0.00000001% (or 1 part in 10 billion).
- **Distance**
 - Distance accuracy is dependent on the geometry types. Point-to-point distances are accurate to the same degree as length calculations. Other distance calculations are generalized for long or short distances, and are accurate to within 0.2% (or 2 parts per thousand).

Geodetic or Projected Coordinate Systems?

- **Geodetic CS are the natural choice for large problems (those where no single projected system can give accurate results).**
- **The generality of geodetic solutions will often be useful, even for small problems.**
- **The most accurate results for small problems will likely be obtained with a suitable projected CS.**

ORACLE

10-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Geodetic or Projected Coordinate Systems?

Geodetic coordinate systems are the natural choice for large problems where no single projected coordinate system can give accurate results.

The generality and ease of use of geodetic solutions will be both practical and useful even for small problems.

The most accurate results for very localized problems will likely be obtained using a suitable projected coordinate system.

Based on your requirements and the accuracy information presented, you can decide which type of coordinate system best suits your needs, geodetic or projected.

User-Defined Coordinate Systems

ORACLE

10-29

Copyright © Oracle Corporation, 2001. All rights reserved.

User-Defined Coordinate Systems

- **Users can create their own coordinate systems based on the ellipsoids, datums, and projections provided by Oracle Spatial**
- **SRID and WKTEXT required**
- **Oracle will provide a unique SRID for each user-defined CS through an escrow mechanism**
 - **This process should be strictly followed to avoid any conflicts between different user-defined coordinate systems**
 - **The procedure for requesting an SRID is not yet finalized**

ORACLE

10-30

Copyright © Oracle Corporation, 2001. All rights reserved.

User-Defined Coordinate Systems

It is now possible for users to create their own coordinate systems in Oracle Spatial. These coordinate systems are based on the ellipsoids, datums, and projections incorporated into Oracle Spatial.

User-defined coordinate systems are important because they allow users to define coordinate systems that are not initially provided by Oracle Spatial. User-defined coordinate systems may be highly localized to a specific region or town.

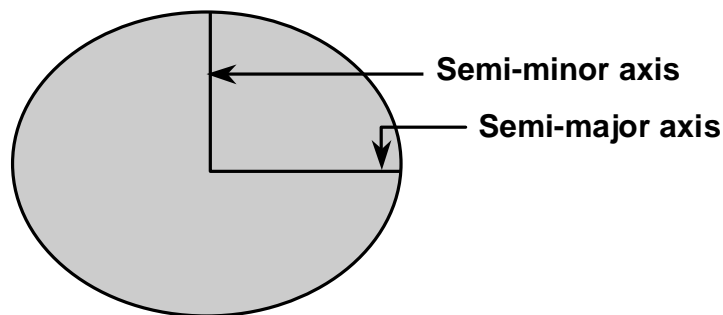
Users will need to have a new SRID value and WKTEXT definition to create a user-defined coordinate system.

To create a user-defined coordinate system, a registration process will be set up to request a unique SRID value from Oracle. The registration process should be followed to ensure that duplicate SRID values are not used. User-defined coordinate systems will have a minimum value of 1,000,000.

The procedure for requesting an SRID is not yet finalized.

Ellipsoids

- Defined by the
 - SEMI_MAJOR_AXIS
 - INVERSE_FLATTENING
- MDSYS.SDO_ELLIPSOIDS lists the supported ellipsoids



ORACLE

10-31

Copyright © Oracle Corporation, 2001. All rights reserved.

Ellipsoids

The shape of the Earth is highly irregular. To make precise calculations on the Earth's surface, the shape of the Earth is approximated to a mathematical figure: an ellipsoid. Ellipsoids are defined by their axis:

The semi-major axis of an ellipsoid is the greater of the two axes.

The semi-minor axis of an ellipsoid is the smaller of the two axes.

Inverse flattening is calculated as $\text{semi-major} / (\text{semi-major} - \text{semi-minor})$

For the surface of the Earth, the inverse flattening of the supported ellipsoids is in the range of 293 to 309.

In Oracle Spatial, the supported ellipsoids are found in the MDSYS.SDO_ELLIPSOIDS table:

```
SQL> describe MDSYS.SDO_ELLIPSOIDS
```

Name	Null?	Type

NAME		VARCHAR2(64)
SEMI_MAJOR_AXIS		NUMBER
INVERSE_FLATTENING		NUMBER

Datums

- A datum is a means of representing the figure of the Earth
- An oblate ellipsoid of revolution that approximates the surface of the Earth is used as the basis for a datum
- This ellipsoid is the reference for the systems of geodetic coordinates
- MDSYS.SDO_DATUMS table lists the supported datums

Datums

Datums tie mathematical figures (ellipsoids) to the surface of the Earth so that horizontal distances can be accurately determined.

In Oracle Spatial, three shift and three rotation parameters adjust the reference ellipsoid to better approximate the region or area-of-interest on the Earth's surface. A scale_adjust parameter can also be used to scale the ellipsoid to better approximate the surface.

In Oracle Spatial, the supported datums are found in the MDSYS.SDO_DATUMS table:

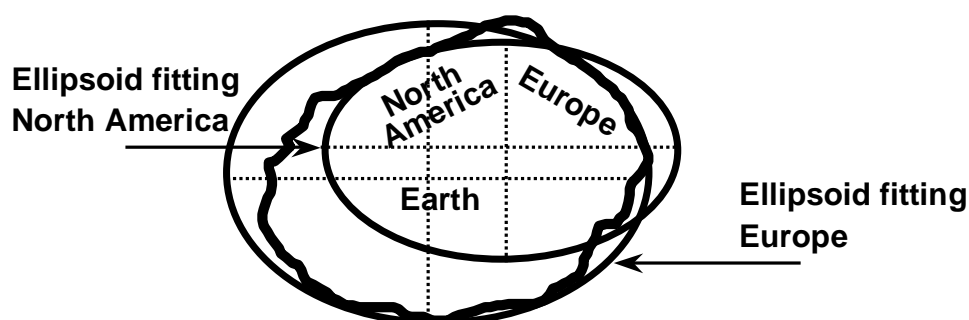
```
SQL> describe MDSYS.SDO_DATUMS
```

Name	Null?	Type

NAME		VARCHAR2(64)
SHIFT_X		NUMBER
SHIFT_Y		NUMBER
SHIFT_Z		NUMBER
ROTATE_X		NUMBER
ROTATE_Y		NUMBER
ROTATE_Z		NUMBER
SCALE_ADJUST		NUMBER

Datums

- Reference systems consisting of an ellipsoid and an origin
- Ellipsoid is chosen to closely conform to the area of interest



ORACLE

10-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Datums (continued)

Datums are reference systems consisting of an ellipsoid and an origin. The reference ellipsoid is chosen so that it closely conforms to the region or area-of-interest on the Earth's surface.

Nine elements define a datum in Oracle Spatial:

- Three elements define the position of the origin
 - SHIFT_X
 - SHIFT_Y
 - SHIFT_Z
- Three elements define the orientation of the ellipsoid
 - ROTATE_X
 - ROTATE_Y
 - ROTATE_Z
- One element defines an optional scale factor
 - SCALE_ADJUST
- Two elements define the reference ellipsoid
 - SEMI_MAJOR_AXIS
 - INVERSE_FLATTENING

Datums (continued)

Two commonly utilized datums include:

The World Geodetic System Datum 1984 (WGS 84): This datum is referenced to the center of the Earth, and is commonly used by Global Positioning Systems (GPS). It is based on the WGS84 reference ellipsoid.

North American Datum 1983 (NAD 83): This datum is commonly used in the North American. It was adopted by the United States in 1983 and is based on the GRS 80 ellipsoid.

Projections

- **Represent a spherical surface on a plane**
- **MDSYS.SDO_PROJECTIONS table lists valid projections**
- **Example projections**
 - **Lambert Conformal Conic**
 - **Transverse Mercator**
 - **Albers Conical Equal Area**

ORACLE

10-35

Copyright © Oracle Corporation, 2001. All rights reserved.

Projections

Projections “flatten” the representation of the Earth’s surface into two dimensions, often for a specific region. It is not possible to map a spherical surface perfectly onto a plane without some distortion. Distortions may alter the shape, scale, area, and direction of a projected region. No single projection can satisfy the requirements for every mapping application. Projections are chosen to maintain certain characteristics or limit distortions in a particular area. For example, conformal projections result in maps showing small areas in their correct shape. Equal-area projections result in maps showing all areas in their proper relative size. In equidistant projections, distances are correctly represented from one central point to other points on the map, and in azimuthal projections, the maps show the correct direction of any point relative to a central point.

Projections (continued)

The MDSYS.SDO_PROJECTIONS table contains over 40 valid projections utilized by Oracle Spatial. A few are listed below:

- Lambert Conformal Conic
 - A conformal projection used by the USGS for mapping states having a large east-west dimension.
- Transverse Mercator
 - A conformal projection used by the USGS for quad maps. Distances are true along the central meridian, and are reasonably accurate within 15 degrees of the central meridian.
- Albers Conical Equal-Area
 - An equal-area projection used by USGS for large area maps. It is also used for mapping regions having large east-west dimensions.

Well-Known Text (WKT) for a Coordinate System

- **WKT defines the coordinate system in Oracle Spatial**
 - **WKTEXT column in the CS_SRS table**
- **Oracle Spatial uses WKT which conforms to the Open GIS Consortium (OGC) standard**
- **The grammar for the WKT is described in the Oracle Spatial User Guide**
 - **This grammar is used for adding user-defined coordinate systems**

ORACLE

10-37

Copyright © Oracle Corporation, 2001. All rights reserved.

Well-Known Text (WKT) for a Coordinate System

Oracle Spatial uses Well-Known Text (WKT) to define coordinate systems within Oracle Spatial. The Well-Known Text is found in the WKTEXT column of the MDSYS.CS_SRS table.

The Well-Known Text in Oracle Spatial conforms to the Open GIS Consortium (OGC) standard, and the exact grammar is described in the Oracle Spatial User's Guide. The grammar as described must be used when adding user-defined coordinate systems.

Well-Known Text Example

- **Geodetic Coordinate System**

```
'GEOGCS [ "Longitude / Latitude (WGS 84)",  
  DATUM ["WGS 84",  
    SPHEROID ["WGS 84", 6378137.000000, 298.257224]],  
  PRIMEM [ "Greenwich", 0.000000 ],  
  UNIT ["Decimal Degree", 0.01745329251994330]]'
```

Well-Known Text Example for a Geodetic Coordinate System

GEOGCS: The prefix for all geodetic coordinate systems is GEOGCS, which stands for GEOGraphic (geodetic) Coordinate System. “Longitude / Latitude (WGS 84)” is the name of the geodetic coordinate system.

DATUM: The datum for this coordinate system is “WGS 84”, which can be found in the MDSYS.SDO_DATUMS table.

SPHEROID: The spheroid or reference ellipsoid “WGS 84”, used by the WGS 84 datum. (WGS 84: semi-major axis = 6378137 meters; inverse flattening = 298.257224).

PRIMEM: The prime meridian “Greenwich”. In this coordinate system the prime meridian runs through Greenwich, UK.

UNIT: The unit “Decimal Degree”, (which is the unit for all geodetic coordinate systems in Oracle Spatial), and “0.01745329251994330” is the conversion factor to radians.

Note: The only angular units allowable in Oracle Spatial are decimal degrees. The table MDSYS.SDO_ANGLE_UNITS has other definitions for future use.

Well-Known Text Example

- **Projected Coordinate System**

```
PROJCS["Wyoming 4901, Eastern Zone (1983, meters)",  
  GEOGCS ["NAD 83 (Continental US)",  
    DATUM ["NAD 83 (Continental US)",  
      SPHEROID ["GRS 80", 6378137.000000, 298.257222]],  
    PRIMEM [ "Greenwich", 0.000000 ],  
    UNIT ["Decimal Degree", 0.01745329251994330]],  
  PROJECTION ["Transverse Mercator"],  
  PARAMETER ["Scale_Factor", 0.999938],  
  PARAMETER ["Central_Meridian", -105.166667],  
  PARAMETER ["Latitude_Of_Origin", 40.500000],  
  PARAMETER ["False_Easting", 200000.000000],  
  UNIT ["Meter", 1.000000000000]
```

1

ORACLE

10-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Well-Known Text Example for a Projected Coordinate System

PROJCS: The prefix for all PROJected Coordinate Systems is PROJCS. “Wyoming 4901, Eastern Zone (1983, meters)” is the name of the projected coordinate system.

GEOGCS: The geographic coordinate system for this projection, which is “NAD 83 (Continental US)”.

DATUM: The datum for the coordinate system, which is “NAD 83 (Continental US)”.

SPHEROID: The spheroid or reference ellipsoid “GRS 80”, used by the NAD 83 datum.

PRIMEM: The prime meridian “Greenwich”. In this coordinate system the prime meridian runs through Greenwich, UK.

UNIT: The unit “Decimal Degree”, (which is the unit for all geodetic coordinate systems in Oracle Spatial), and “0.01745329251994330” is the conversion factor to radians.

PROJECTION- The type of projection “Transverse Mercator”, which defines the conversion (or mapping) from longitude/latitude to X,Y coordinates.

Scale_Factor: The factor that is applied to the formula that transforms Long/Lat to the projected X,Y values.

Central_Meridian and Latitude_Of_Origin: These define the center of the projection. The center of the projection is where the projection is most accurate.

False_Easting (and False_Northing when defined): These help ensure that X,Y values in the projected coordinate system are always positive.

UNIT: The unit “Meter”, which is the unit for the projected coordinates, and “1.000000000000” is the conversion factor.

Creating a User-Defined Coordinate System

- **Pick an ellipsoid**
 - Can come from the **MDSYS.SDO_ELLIPSOIDS** table
 - Can create a new ellipsoid, but cannot duplicate name
- **Pick a datum**
 - Can reuse a provided datum (over 100 supported)
 - **MDSYS.SDO_DATUMS** table
 - Can create new datum, but cannot duplicate datum names
 - Include shift, rotation, and scale parameters in WKT if any parameter is not zero
- **When defining a projected coordinate system**
 - Pick a projection from the supported projections
 - Over 40 supported
 - **MDSYS.SDO_PROJECTIONS** table

ORACLE

10-40

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a User-Defined Coordinate System

To create a new user-defined coordinate system a user must choose an ellipsoid, and a datum. If the coordinate system is projected, a user must also choose a projection.

The ellipsoid can come from the **MDSYS.SDO_ELLIPSOIDS** table, or the user can create their own ellipsoid. If the user creates their own ellipsoid, it cannot have the same name as any ellipsoid in the **MDSYS.SDO_ELLIPSOIDS** table.

The datum used can be an existing datum (from the **MDSYS.SDO_DATUMS** table), or the user can create their own datum. If the user decides to create their own datum they must create a new datum name. Duplicate datum names are not allowed. Also, if the user decides to create their own datum, the seven parameters for translation, scale and rotation are applied to the ellipsoid used for the datum. The default for all of the parameters is zero (0). If any of the seven parameters are not zero, then they all must be specified.

The projection, if required, must come from the **MDSYS.SDO_PROJECTIONS** table. There are over 40 projections to choose from.

Creating a User-Defined Coordinate System

Use SQL DML (insert/update/delete) statements to add, modify, or delete coordinate systems entries in the MDSYS.CS_SRS table

- **Log in as MDSYS**
- **Be careful to modify only the Coordinate Systems that you add**

ORACLE

10-41

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a User-Defined Coordinate System (continued)

After the well-known text is formed, an SRID value is assigned, and the name of the user-defined coordinate system is decided. Use a simple SQL DML statement to add the the coordinate system entry into the MDSYS.CS_SRS table. To insert/update/delete data in the MDSYS.CS_SRS table, log into Oracle as MDSYS. *Users should exercise caution to ensure they only modify or delete coordinate systems which they have added.*

User-Defined Coordinate Systems Example

Defining a new CS with new projection parameters (UTM)

```
insert into cs_srs values ('UTM Zone 44.5, Northern Hemisphere (WGS 84)',  
    1082378, 1082378, 'Oracle',  
    'PROJCS["UTM Zone 44.5, Northern Hemisphere (WGS 84)",  
    GEOGCS [ "WGS 84",  
    DATUM [ "WGS 84 ",  
    SPHEROID [ "WGS 84", 6378137.000000, 298.257224]],  
    PRIMEM [ "Greenwich", 0.000000 ],  
    UNIT [ "Decimal Degree", 0.01745329251994330]],  
    PROJECTION [ "Transverse Mercator",  
    PARAMETER [ "Scale_Factor", 0.999600],  
    PARAMETER [ "Central_Meridian", 84.000000],  
    PARAMETER [ "False_Easting", 500000.000000],  
    UNIT [ "Meter", 1.000000000000]]', NULL);  
  
('UTM Zone 44, Northern Hemisphere (WGS 84)', 82378, 82378, 'Oracle',  
    'PROJCS["UTM Zone 44 Northern Hemisphere (WGS 84)",  
    GEOGCS [ "WGS 84",  
    DATUM [ "WGS 84 ",  
    SPHEROID [ "WGS 84", 6378137.000000, 298.257224]],  
    PRIMEM [ "Greenwich", 0.000000 ],  
    UNIT [ "Decimal Degree", 0.01745329251994330]],  
    PROJECTION [ "Transverse Mercator",  
    PARAMETER [ "Scale_Factor", 0.999600],  
    PARAMETER [ "Central_Meridian", 81.000000],  
    PARAMETER [ "False_Easting", 500000.000000],  
    UNIT [ "Meter", 1.000000000000]]')
```

ORACLE

User-Defined Coordinate Systems Example

This example shows an insert statement used to add a user-defined coordinate system into Oracle Spatial, and the values associated with a similar existing coordinate system.

Note that the Coordinate System names are slightly different and that the SRID values are different.

The only changes in the WKT between the user-defined coordinate system and the existing coordinate system are:

The name change (to UTM Zone 44.5, Northern Hemisphere (WGS 84), and

The central meridian has shifted from 81 degrees to 84 degrees.

There are UTM zones defined every 6 degrees. In this example the user's data may have been centered around 84 degrees longitude so a new coordinate system was defined for more accurate answers in the zone around 84 degrees.

User-Defined Coordinate System Example

Defining a new coordinate system with a new ellipsoid and a datum shift

```
-- use the number 1008307 for the new datum
delete from cs_srs where srid = 1008307;
insert into cs_srs values
( 'Longitude / Latitude (WGS 90)', 1008307, 1008307, 'Oracle',
'GEOGCS [ "Longitude / Latitude (WGS 90)",
DATUM [ "WGS 90",
SPHEROID [ "WGS 90", 6378137.032499, 298.257236], 100, 100, 0, 0, 0,0, 0],
PRIMEM [ "Greenwich", 0.000000 ],
UNIT [ "Decimal Degree", 0.01745329251994330]]',NULL);

-- compare this to the WKT for WGS 84
( 'Longitude / Latitude (WGS 84)', 8307, 8307, 'Oracle',
'GEOGCS [ "Longitude / Latitude (WGS 84)",
DATUM [ "WGS 84",
SPHEROID [ "WGS 84", 6378137.000000, 298.257224]],
PRIMEM [ "Greenwich", 0.000000 ],
UNIT [ "Decimal Degree", 0.01745329251994330]]',NULL);
```

ORACLE

10-43

Copyright © Oracle Corporation, 2001. All rights reserved.

User-Defined Coordinate System Example (continued)

In this example a new coordinate system with a datum called WGS 90 is defined. The new coordinate system is similar to WGS 84. The WGS 90 coordinate system includes a new ellipsoid which is named WGS 90. This new ellipsoid name cannot match any of the existing ellipsoid names in Oracle Spatial. The difference between the semi-major axis of the WGS 90 ellipsoid and the WGS 84 ellipsoid is .032499 meters. The difference in the inverse flattening between the two ellipsoids is also very small. The WGS 90 coordinate system also includes a new datum called WGS 90. The new datum is applied to ellipsoid WGS 90, and includes shift parameters of +100 meters along each of the X and Y axes. All of the shift, rotation, and scale parameters must appear in a user-defined datum if any of the parameters are not zero. For Oracle-defined coordinate systems, the datum shift, rotation, and scale parameters do not appear in the definition of the DATUM within the WKT; they appear in the MDSYS.SDO_DATUMS table.

Currently users cannot insert into the MDSYS.SDO_DATUMS table, so the datum is defined as above, with the X,Y, and Z shift, X,Y, and Z rotation, and the scale factor all within the DATUM section (following the SPHEROID) of the WKT.

Local Coordinate Systems

ORACLE

10-44

Copyright © Oracle Corporation, 2001. All rights reserved.

Local Coordinate Systems

- **Non-Earth Cartesian coordinate systems**
- **Use a local coordinate system when no geographic reference is required**
 - CAD systems, building drawings, field survey
- **Allows Oracle Spatial to do unit conversions of geometries**
- **Many units supported including millimeter, centimeter, meter, kilometer, mile, nautical mile, foot, inch, chain, link, and so on**

ORACLE

10-45

Copyright © Oracle Corporation, 2001. All rights reserved.

Local Coordinate Systems

Local coordinate systems in Oracle Spatial are non-Earth Cartesian coordinate systems. They are useful when no geographic reference is required. For example, CAD systems and architectural drawings will not change relative to where they are located on the surface of the Earth. An engine part that needs to be 1 cm in size and positioned relative to other engine parts will be the same size and have the same relative location no matter where on the Earth's surface the engine is manufactured.

Oracle Spatial can be used to convert between the various local coordinate systems supported.

There is local coordinate system support with many different units, including millimeter, centimeter, meter, kilometer, mile, nautical mile, foot, inch, yard, chain, link, rod, fathom, and so on.

Local CS Examples

- **Non-Earth (Meter), 262144, 262144, Oracle,**
LOCAL_CS ["Non-Earth (Meter)",
LOCAL_DATUM ["Local Datum", 0],
UNIT ["Meter", 1.0],
AXIS ["X", EAST],
AXIS ["Y", NORTH]]
- **Non-Earth (Kilometer), 262145, 262145, Oracle,**
LOCAL_CS ["Non-Earth (Kilometer) ",
LOCAL_DATUM ["Local Datum", 0],
UNIT ["Kilometer", 1000.0],
AXIS ["X", EAST],
AXIS ["Y", NORTH]]

ORACLE

10-46

Copyright © Oracle Corporation, 2001. All rights reserved.

Local Coordinate Systems Example

All of the local coordinate systems generally look similar. However, they have different SRIDs and the WKTEXT differs.

LOCAL_CS: The prefix for all local coordinate systems is LOCAL_CS. The WKTEXT always starts with LOCAL_CS and the name.

LOCAL_DATUM: The local datum for local coordinate systems always has “Local Datum” and “0”.

UNIT: The unit is the only part of the WKTEXT besides the name of the local coordinate system that changes. The unit has the name of the units, and the conversion factor from meter.

Summary

In this lesson, you should have learned:

- **The benefits of the whole Earth geometry model**
- **How and when to implement coordinate systems transformations**
- **The process for creating user defined coordinate systems**
- **About local coordinate systems**

ORACLE

11

Advanced Spatial Indexing Concepts

ORACLE

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Understand and create partitioned spatial indexes**
- **Understand and create function-based indexes**
- **Understand how to embed and index spatial objects within objects**

ORACLE

Spatial Index Partitioning

ORACLE

11-3

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle Table Partitioning and Oracle Spatial

- Oracle table partitioning is not unique to Oracle Spatial
- Before Oracle9i Spatial local spatial indexes on partitioned tables were not supported
- In Oracle9i Spatial, local spatial indexes on partitioned tables are supported

ORACLE

11-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle Table Partitioning and Oracle Spatial

Oracle table partitioning is an Oracle Database feature, and is not specific to Oracle Spatial. Table partitioning allows users to break tables up into smaller pieces called partitions based on a value called a partition key. The value of the partition key determines which of the partitions each row is stored in. Queries are written to use a table, and Oracle will determine which partitions it needs to search. For query performance, Oracle table partitioning is only beneficial when the partition key is specified in the WHERE clause.

Before Oracle9i local (or partitioned) spatial indexes on partitioned tables were not supported.

As of Oracle9i local (or partitioned) spatial indexes on partitioned data are supported.

Why Partition Spatial Indexes?

- **Performance:**
 - Reduce response times for long-running queries: local index scan
 - Reduce response times for concurrent queries: parallel I/O on each partition
- **Maintenance:**
 - Partition-level index rebuilds
 - Partition-level table archives

ORACLE

11-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Why Partition Spatial Indexes?

There are several reasons users might want to partition spatial indexes. Some performance reasons for partitioning spatial indexes include:

- The time to execute long-running queries can be reduced by only examining indexes associated with necessary partitions.
- The time to execute queries run in parallel can be reduced because:
 - Examining the index associated with the necessary partition is faster.
 - I/O contention can be reduced because separate partitions can be examined concurrently.

Long-term administration and maintenance of spatial indexes is easier:

- The spatial index associated with a single partition can be rebuilt.
- The spatial index associated with a single partition can be archived or restored.

Oracle9i Spatial Index Partitioning

Addresses problem of supporting very large tables and indexes

- **Decomposes tables and indexes into smaller pieces called partitions**
- **SQL statements can access and manipulate the partitions rather than entire tables**
- **Table partitions and/or index partitions can be mapped to different tablespaces**

ORACLE

11-6

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle9i Spatial Index Partitioning

Partitioning helps address many of the problems associated with supporting very large tables and indexes. Tables and indexes are broken up into smaller pieces called partitions. SQL statements can then access and manipulate data on a partition-by-partition basis rather than the entire tables and indexes. Tables partitions and/or index partitions can be mapped to different tablespaces and/or hardware controllers.

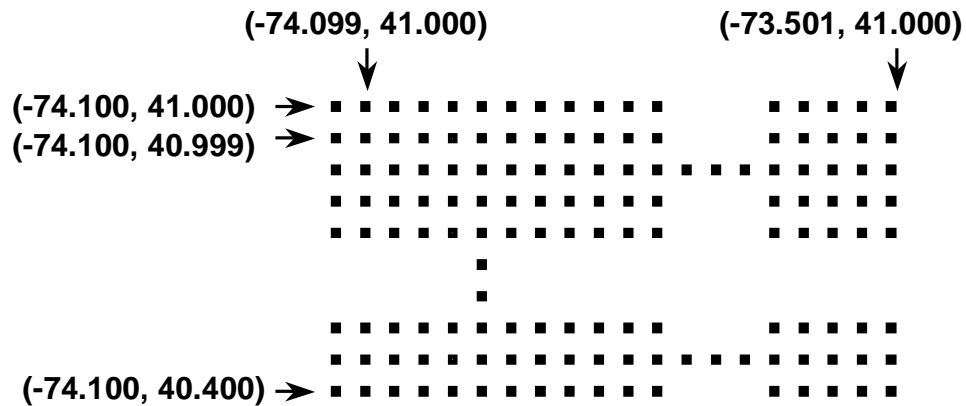
Data Set Description for Partitioned Spatial Index Examples

ORACLE

11-7

Copyright © Oracle Corporation, 2001. All rights reserved.

Fabricated Yellow Pages Data (Businesses in New York)



- All business locations stored in longitude/latitude
- Businesses on the same latitude are 0.001 decimal degrees apart
- Latitudes increase by 0.001 decimal degrees
- 360,600 businesses in table

ORACLE

Fabricated Yellow Pages Data (Businesses in New York)

Some yellow pages data has been fabricated to demonstrate Oracle9i partitioned spatial indexes. All business locations are stored in longitude/latitude (WGS 84).

Businesses on the same latitude are 0.001 decimal degrees apart. Latitudes also increase by 0.001 decimal degrees.

There are a total of 360,600 businesses in this fabricated data set. The longitude/latitude values chosen for the businesses are in the vicinity of New York City.

The YELLOW_PAGES Table

ID	Name	Category	Location
1	ITALIAN RESTAURANT 1	1	(-74.100, 40.400)
2	CHASE BANK 1	2	(-74.100, 40.401)
3	MOBIL GAS 1	3	(-74.100, 40.402)
4	PATH MARK GROCERY 1	4	(-74.100, 40.403)
5	SHERATON HOTEL 1	5	(-74.100, 40.404)
6	FORD AUTOS 1	6	(-74.100, 40.405)
7	THAI RESTAURANT 1	1	(-74.100, 40.406)
25	ITALIAN RESTAURANT 2	1	(-74.100, 40.424)

Six categories defined that correspond to:

- 1 for restaurants
- 2 for banks
- 3 for gas stations
- 4 for grocery stores
- 5 for hotels
- 6 for auto dealers

ORACLE

11-9

Copyright © Oracle Corporation, 2001. All rights reserved.

The Yellow Pages Table

The table has the following four columns:

- **ID:** This is a unique ID associated with each business
- **Category:** Each business is assigned to a category. There are six possible values for category (1-6). The value of category corresponds to:
 - 1 for restaurants
 - 2 for banks
 - 3 for gas stations
 - 4 for grocery stores
 - 5 for hotels
 - 6 for auto dealers
- **Name:** This is a name associated with each business. In this data set, a name is never repeated. You can see that the businesses with ID=1 and ID=25 are both Italian restaurants, but a unique number is appended to the name to differentiate the two by name.
- **Location:** This is an MDSYS.SDO_GEOMETRY column, and contains the longitude/latitude values associated with the business.

Creating a Partitioned Spatial Index

ORACLE

11-10

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Partitioning Parameters

Two types of parameters

- **Spatial parameters**
 - These parameters define the type of index created (R-tree, quadtree, level, and so on)
 - Must be the same for all partitions
- **Storage parameters**
 - These can (and should) be different for each partition
 - These parameters help reduce the I/O conflict between different partitions

ORACLE

11-11

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Partitioning Parameters

There are two types of parameters associated with partitioned spatial indexes.

Spatial parameters are the parameters that define the type of index (quadtree or R-tree) as well as the individual parameters relating to the type of spatial index (for example, SDO_LEVEL, SDO_INDX_DIMS, and so on) as well as generic spatial-related parameters (such as LAYER_GTYPE). Spatial parameters must remain the same across all index partitions associated with a table.

Storage parameters are parameters relating where and how the index data is stored. These are parameters that can (and perhaps should) be different for each partition. For example, specifying a different tablespace parameter for each partition would reduce I/O conflicts between different partitions.

Oracle9i Spatial Index Partitioning

- Only range partitioning of base table supported
 - Not hash or composite
 - Extensible indexing limitation
 - Partition key cannot be the SDO_GEOMETRY column
- Choose a partition key that will:
 - Always be in the WHERE clause
 - Create meaningful partitions when geometries are separated based on the partition key value
- With an effective partition key, performance gains attributed to partitioning will be greater on larger tables

```
CREATE TABLE yellow_pages_part (id      NUMBER,  
                                name     VARCHAR2(50),  
                                category  NUMBER  
                                location  MDSYS.SDO_GEOMETRY)  
PARTITION BY RANGE (category)  
(PARTITION p1 VALUES LESS THAN (2),  
 PARTITION p2 VALUES LESS THAN (3),  
 PARTITION p3 VALUES LESS THAN (4),  
 PARTITION p4 VALUES LESS THAN (5),  
 PARTITION p5 VALUES LESS THAN (6),  
 PARTITION p6 VALUES LESS THAN (MAXVALUE));
```

ORACLE

11-12

Copyright © Oracle Corporation, 2001. All rights reserved.

Oracle9i Spatial Index Partitioning

Oracle Spatial index partitioning can only be used if the base table is partitioned by range. Range partitions are populated according to the value in the partition key column. Valid range partition key columns must contain scalar values, such as number, text, or date. Raw, LOB, BLOB or object type columns are not valid range partition keys..

Hash (based on a hash function applied to the values in a column) and composite partitioning (where first a range partition is done, then a hash partition is done) are not currently supported for spatial partitioned indexes.

Based on the restrictions noted previously, a partition key cannot be of type SDO_GEOMETRY.

Oracle9i Spatial Index Partitioning (continued)

The example shows the creation of a partitioned table called YELLOW_PAGES_PART. YELLOW_PAGES_PART is partitioned by range on column CATEGORY, and has six partitions named P1 through P6. The following describes which rows are stored in each partition:

- P1 contains rows for businesses that have a category value less than 2.
- P2 contains rows for businesses that have a category value less than 3.
- P3 contains rows for businesses that have a category value less than 4.
- P4 contains rows for businesses that have a category value less than 5.
- P5 contains rows for businesses that have a category value less than 6.
- P6 contains rows for businesses that have a category value greater than or equal to 6.

The partition key must be present in the WHERE clause for performance benefits attributed to partitioning. Also, choose a partition key that groups your geometries into partitions in a meaningful way. In this example, the data is partitioned by CATEGORY. Placing CATEGORY in the WHERE clause ensures a spatial operator will only search index entries for geometries for that category.

Spatial Index Partitioning Syntax

```
CREATE INDEX yp_part_sidx ON yellow_pages_part (location)
INDEXTYPE IS MDSYS.SPATIAL_INDEX PARAMETERS ('SDO_LEVEL=9')
LOCAL
(PARTITION IP1 PARAMETERS ('TABLESPACE=TBS_1 SDO_LEVEL=9'),
 PARTITION IP2 PARAMETERS ('TABLESPACE=TBS_2 SDO_LEVEL=9'),
 PARTITION IP3,
 PARTITION IP4,
 PARTITION IP5,
 PARTITION IP6);
```

- The **LOCAL** keyword is *required* to create a partitioned index

Note: The type of index, and **SDO_LEVEL** if a quadtree index, must be the same for all partitions

ORACLE

11-14

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Partitioning Syntax

The syntax to create a partitioned spatial index on a partitioned table is very similar to the syntax required to create a spatial index. The syntax starts identically. Where the nonpartitioned create index syntax ends the keyword **LOCAL** is specified, then partitioning information is specified on a per partition basis.

The parameters works the same as the parameters in the **CREATE INDEX** statement. In the example given, the index partitions are named **IP1** through **IP6**, corresponding to index partitions **P1** through **P6**.

Note that initial partitioning information comes from the first **PARAMETERS** clause in the create index statement. Each of the **PARAMETERS** clauses for the local partitions overrides the initial partitioning information. If a partition's **PARAMETERS** clause is omitted, then the index parameters in the first **PARAMETERS** clause are used.

Note that the **PARAMETERS** clause for each partition can only be used to modify storage parameters and not to modify spatial index parameters.

In the example, **SDO_LEVEL** is repeated in each partition's **PARAMETERS** clause because if it were omitted that partition would try to create an R-tree index. This would violate the rule that all partitions must have the same spatial index parameters.

Spatial Index Partitioning Syntax

In this example, some default and some specified values are used for the placement of index partitions

- **For each table partition, a corresponding index partition is created:**
 - 6 table partitions
 - 6 index partitions
- **Index partition IP1 is placed in the TBS_1 tablespace**
- **Index partition IP2 is placed in the TBS_2 tablespace**
- **Index partitions IP3-IP6 are in DEFAULT tablespace**

ORACLE

11-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Partitioning Syntax (continued)

In the previous example the use of both default and specified values is shown for the placement of index partitions.

For each table partition a corresponding index partition is created. Table partition P1 in the example corresponds to index partition IP1. For the table, there are six table partitions and six index partitions.

From the parameters clause that follows the index partition names, it can be seen that index partition IP1 is stored in the TBS_1 tablespace. Index partition IP2 is stored in the TBS2 tablespace. Because the PARAMETERS clause was not specified for IP3, IP4, IP5, and IP6, those index partitions are stored in the DEFAULT tablespace for the user.

Spatial Index Partitioning Restrictions

- **Partition key must be a scalar value**
- **Hash and composite partitioning are not supported**
- **Spatial index-specific parameters (for example, index type, quadtree or R-tree) must be the same across all partitions**
 - If quadtree, SDO_LEVEL must be the same in all partitions
- **Some ALTER TABLE functionality is disabled**
 - Exchange
- **Can split and merge table partitions**
 - Must rebuild associated index partitions

ORACLE

11-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Spatial Index Partitioning Restrictions

Oracle Spatial index partitioning can only be used if the base table is partitioned by range. Range partitions are populated according to the value in the partition key column. Valid range partition key columns must contain scalar values, such as number, text, or date. Raw, LOB, BLOB or object type columns are not valid range partition keys..

Hash (based on a hash function applied to the values in a column) and composite partitioning (where first a range partition is done, then a hash partition is done) are not supported for spatial partitioned indexes.

The spatial index-specific parameters must be the same for each partition - only storage parameters can change.

The exchange partition alter table functionality is disabled with partitioned spatial indexes. Exchange partition exchanges data and indexes from a nonpartitioned table with the data and indexes from one partition of a partitioned table.

The split and merge partition functionality is supported with partitioned spatial indexes, but the index associated with the partition must be rebuilt. Split partition takes a single partition and subdivides it based on finer granularity associated with the partitioning key. Merge partition takes multiple partitions and merges them together so there are fewer partitions. When partitions are merged the partitioning key encompasses more data.

Query Examples Using Partitioned and Nonpartitioned Tables

ORACLE

11-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Nonpartitioned Example

- How many restaurants are within 8 miles of a given location?

```
-- non-partitioned case
SELECT count(*)
FROM yellow_pages a
WHERE sdo_within_distance (a.location,
        mdsys.sdo_geometry (2001, 8307,
        mdsys.sdo_point_type (-73.8, 40.7, null), null, null),
        'distance=8 unit=mile') = 'TRUE'
AND category = 1;
-- runs in 4.32 seconds
```

```
-- non-partitioned case
SELECT /*+ no_index (a yp_category_idx) */ count(*)
FROM yellow_pages a
WHERE sdo_within_distance (a.location,
        mdsys.sdo_geometry (2001, 8307,
        mdsys.sdo_point_type (-73.8, 40.7, null), null, null),
        'distance=8 unit=mile') = 'TRUE'
AND category = 1;
-- runs in 2.14 seconds
```

- **Note: Sometimes, no_index hints can help performance, irrespective of partitioning.**

ORACLE

Nonpartitioned Example

Both queries in this slide are answering the following question, “How many restaurants are within eight miles of a given location?”

Both queries are executed using a nonpartitioned yellow pages table. The nonpartitioned table has an index on CATEGORY (yp_category_idx) and a spatial index on LOCATION. Category 1 corresponds to restaurants.

Note the second query ran faster when the yp_category_idx index is disabled with the no_index hint. Sometimes it is faster to reduce the selected set returned from a spatial operator without using other indexes. This holds true for nonpartitioned as well as partitioned tables.

Keep in mind when joining to other tables, nonspatial index are usually crucial.

Partitioned Example

- How many restaurants are within 8 miles of a given location?

```
-- partitioned case

SELECT count(*)
FROM yellow_pages_part a
WHERE sdo_within_distance (a.location,
    mdsys.sdo_geometry (2001, 8307,
        mdsys.sdo_point_type (-73.8, 40.7, null), null, null),
    'distance=8 unit=mile') = 'TRUE'
AND category = 1;

-- runs in .43 seconds
```

ORACLE

11-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Partitioned Example

This query answers the same question posed in the previous slide, “How many restaurants are within eight miles of a given location?”.

This query is executed against the partitioned yellow pages table and the partition key (CATEGORY) is in the where clause. This query ran faster than the queries against the nonpartitioned table in the previous slide.

Partitioned Example: SDO_NN

- Find the three closest businesses (any kind) to a given location
 - SDO_NN executes once per partition
 - This example does not include a partition key, so all six partitions are searched
 - This example sets 'SDO_NUM_RES=3', but returns 18 rows (three rows per partition)
 - Next slide will show how to answer the question above (that is, only return three rows)

```
-- This query does not answer the question above
-- (returns more than three rows)

SELECT name, sdo_nn_distance(1) distance
FROM yellow_pages_part a
WHERE sdo_nn (a.location,
             mdsys.sdo_geometry (2001, 8307,
                                 mdsys.sdo_point_type (-73.8042, 40.7613, null),
                                 null, null),
             'sdo_num_res=3 unit=mile', 1) = 'TRUE'
ORDER BY distance;
```

ORACLE

11-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Partitioned Example: SDO_NN

The requirement posed in this slide is to find the three closest businesses to a given location. The yellow pages table is partitioned by business category, but this query can not benefit from the partition key (all partitions must be searched).

The SDO_NN operator executes once per partition.

The query shown in the slide sets 'SDO_NUM_RES=3', but returns 18 rows (3 rows X 6 partitions). This query does not meet the requirement specified, which is to find the three closest businesses to a given location.

The next slide will show how to return the three closest businesses.

Partitioned Example: SDO_NN

- Find the three closest businesses (any kind) to a given location
 - Example below ensures only three rows are returned
 - When more than one partition is searched, and a specific number of results are desired, SDO_NN must use an inline view with “rownum < (some_value)” outside of the inline view
 - In the example below, the inline view returns 18 rows (three rows per partition) and “rownum < 4” reduces the result set to three rows

```
-- partitioned table - query does not include partition key

SELECT name, distance
FROM (SELECT name, sdo_nn_distance(1) distance
      FROM yellow_pages_part a
      WHERE sdo_nn (a.location,
                    mdsys.sdo_geometry (2001, 8307,
                                         mdsys.sdo_point_type (-73.8042, 40.7613,null),
                                         null, null),
                    'sdo_num_res=3 unit=mile', 1) = 'TRUE'
      ORDER BY distance)
WHERE rownum < 4;
```

ORACLE

11-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Partitioned Example: SDO_NN (continued)

This query will return the three closest business to the specified point.

When more than one partition is searched, and a specific number of results are desired, SDO_NN must use an inline view with “rownum < (some_value)” outside of the inline view.

In this query, the inline view contains the SQL statement shown on the previous slide (that returns 18 rows), and the result is constrained to 3 rows by specifying “WHERE rownum < 4” outside of the inline view.

Partitioned Example: SDO_NN

Find the three closest restaurants to a given location

- Only looking for restaurants, include the restaurant category partition key
- Restaurant category partition key ensures only one partition is searched, inline view is not necessary
- This query will return the three closest restaurants

```
-- partitioned table - query includes partition key and
-- use of the inline view is not necessary

SELECT name, sdo_nn_distance(1) distance
FROM yellow_pages_part a
WHERE a.category = 1
      AND sdo_nn (a.location,
                  mdsys.sdo_geometry (2001, 8307,
                                      mdsys.sdo_point_type (-73.8042, 40.7613, null), null, null),
                  'sdo_num_res=3 unit=mile', 1) = 'TRUE'
ORDER BY distance;
```

ORACLE

11-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Partitioned Example: SDO_NN (continued)

This query will return the three closest restaurants to a specified point.

Because restaurants correspond to a partition key value (category = 1), this query can be narrowed down to search a single partition. Because a single partition is searched, no in-line view is necessary to ensure 'SDO_NUM_RES' results are returned.

Partitioned Example: SDO_NN

- Find the three closest Italian restaurants to a given location
 - Only looking for restaurants, include the restaurant category partition key
 - Restaurant category partition key ensures only one partition is searched
 - This query will return the three closest Italian restaurants

```
-- partitioned table - query includes the partition key
-- and uses sdo_batch_size instead of sdo_num_res

SELECT name, sdo_nn_distance(1) distance
FROM yellow_pages_part a
WHERE a.category = 1
      AND sdo_nn (a.location,
                  mdsys.sdo_geometry (2001, 8307,
                                      mdsys.sdo_point_type (-73.8042, 40.7613, null),
                                      null, null),
                  'sdo_batch_size=10 unit=mile', 1) = 'TRUE'
      AND name like '%ITALIAN%'
      AND rownum < 4
ORDER BY distance;
```

ORACLE

11-23

Copyright © Oracle Corporation, 2001. All rights reserved.

Partitioned Example: SDO_NN (continued)

This query will return the three closest Italian restaurants to a specified point.

Because restaurants correspond to a partition key value (category = 1), this query can be narrowed down to search a single partition.

As discussed in the spatial query lesson where SDO_NN was introduced, to locate nearest neighbors and also constrain by other attributes (i.e. name like '%ITALIAN%'), SDO_BATCH_SIZE must be used (not SDO_NUM_RES).

As discussed in the spatial query lesson, it is OK to have a rownum < (some_value) followed by an ORDER BY distance clause when using SDO_NN. Traditionally a rownum < (some_value) followed by an ORDER BY will not guarantee correct results, but SDO_NN returns candidates in distance order to the WHERE clause, and correct results are guaranteed.

Function-Based Indexes

ORACLE

11-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Function-Based Indexes

- **Before Oracle9i Spatial, all spatial operations required an SDO_GEOMETRY column in a table**
- **Function-based indexes allow indexes to be built on the result of a function that returns a geometry**
- **Powerful mechanism to enable Oracle Spatial functionality without an SDO_GEOMETRY column in the table**

Function-Based Indexes

Function-based indexes are not unique to Oracle Spatial, and are not new to Oracle9i. For example, a table called EMPLOYEES might have a column called LAST_NAME of type VARCHAR2(30). An index on LAST_NAME can help to quickly find all of the employees with LAST_NAME='SMITH'. If an application requires that a search be done on LAST_NAME starting with 'SMI' (where searches are required to only look at the first three characters of the last name), for example, SUBSTR(LAST_NAME,1,3)='SMI', the application would not utilize the index on LAST_NAME. To overcome this, a function-based index can be created on SUBSTR(LAST_NAME,1,3).

In pre-Oracle9i versions of Oracle Spatial, all spatial operators (that is, SDO_FILTER, SDO_RELATE, SDO_NN, and SDO_WITHIN_DISTANCE) required a column of type MDSYS.SDO_GEOMETRY in a table.

In Oracle9i, Oracle Spatial can benefit from function-based indexes. Function-based indexes allow users to create spatial indexes on a function that returns an MDSYS.SDO_GEOMETRY object. Spatial operators can search function-based indexes as well as traditional spatial indexes.

Having function-based indexes is an extremely powerful feature which gives users the ability to have complete Oracle Spatial functionality with no column of type MDSYS.SDO_GEOMETRY in the table. This is important for many users where an existing table includes location information stored without an MDSYS.SDO_GEOMETRY type and no changes are allowed to the table.

Function-Based Indexes

- **Anticipated common usage for tables with longitude and latitude columns**
- **No requirement to:**
 - Add an **SDO_GEOMETRY** column to the base table
 - Convert all long/lat values to populate an **SDO_GEOMETRY** column

ORACLE

11-26

Copyright © Oracle Corporation, 2001. All rights reserved.

Function-Based Indexes (continued)

An anticipated common usage is for legacy tables that have individual longitude and latitude columns that represent point locations, instead of an `MDSYS.SDO_GEOMETRY` column. For this scenario, a function can be written that puts the longitude/latitude data into the `MDSYS.SDO_GEOMETRY` type and returns it as a geometry object.

There is no need to add and populate an `MDSYS.SDO_GEOMETRY` column in the legacy table.

Function-Based Indexes Procedure

- **Create the function that returns an SDO_GEOMETRY object**
 - The function must be declared as DETERMINISTIC
- **Insert metadata into the USER_SDO_GEOM_METADATA view**
- **Create the spatial index**
- **Perform spatial queries on the data**

ORACLE

11-27

Copyright © Oracle Corporation, 2001. All rights reserved.

Function-Based Indexes Procedure

There are four basic steps required to use function-based indexes.

- A function must be created that returns an MDSYS.SDO_GEOMETRY object. The function must be declared DETERMINISTIC.
- Metadata must be inserted into the USER_SDO_GEOM_METADATA view.
- A spatial index is created on the function.
- Spatial operators can search the function-based spatial index.

Function-Based Indexes Example

```
-- Function on simple relational columns
CREATE OR REPLACE FUNCTION get_geom (
  longitude IN    NUMBER,
  latitude  IN    NUMBER )
RETURN mdsys.sdo_geometry
DETERMINISTIC IS
BEGIN
  RETURN mdsys.sdo_geometry (2001,8307,
    mdsys.sdo_point_type (longitude, latitude, NULL),
    NULL, NULL );
END;
/
CREATE TABLE cities_xy(
  City      varchar2(20),
  State_Abrv varchar2(2),
  Pop90     number,
  Rank90    number,
  longitude  number,
  latitude   number);
```

ORACLE

11-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Function-Based Indexes Example

The example given is a step-by-step implementation of function-based indexes.

In the first step, a function is created that take two input arguments, a longitude and a latitude value which are both of type NUMBER. The function returns a value of type MDSYS.SDO_GEOMETRY. Note the function is declared as deterministic, which is a requirement of function-based indexes. The function simply returns an MDSYS.SDO_GEOMETRY constructor with the following settings:

- SDO_GTYPE=2001 (always a two-dimensional point)
- SDO_SRID = 8307 (which is WGS 84)
- SDO_POINT constructor populated with the longitude and latitude values passed in
- SDO_ELEM_INFO = NULL
- SDO_ORDINATE_ARRAY = NULL

The next step shown is a simple create table statement used to create the base table the function will execute against.

Function-Based Indexes Example

```
INSERT INTO user_sdo_geom_metadata values
('CITIES_XY',
 'SCOTT.GET_GEOM(LONGITUDE,LATITUDE)',
 mdsys.sdo_dim_array(
 mdsys.sdo_dim_element('LONG', -180, 180, 0.5),
 mdsys.sdo_dim_element('LAT', -90, 90, 0.5)),
 8307);

CREATE INDEX cities_func_sidx on
cities_xy(get_geom(longitude,latitude))
INDEXTYPE IS mdsys.spatial_index;

SELECT a.city
FROM cities_xy a
WHERE sdo_within_distance(
    get_geom(longitude,latitude),
    mdsys.sdo_geometry(2001,8307,
        mdsys.sdo_point_type(-73,42,NULL),NULL,NULL),
    'distance = 100 unit = MILE') = 'TRUE';
```

ORACLE

11-29

Copyright © Oracle Corporation, 2001. All rights reserved.

Function-Based Indexes Example (continued)

Continuing the previous example, a record is inserted into the USER_SDO_GEOM_METADATA view. The first column is the table name (CITIES_XY), the second column (the COLUMN_NAME field) has the name of the function with the arguments. The arguments are the LONGITUDE and LATITUDE columns of the CITIES_XY table. Note the name of the function is prefixed by the owner of the function. This is required to differentiate the function GET_GEOM from other functions named GET_GEOM that the user might have access to. The rest of the entry in USER_SDO_GEOM_METADATA is exactly the same as has been discussed previously.

Next, the index is created on the results of the function. The only difference in the statement to create the spatial index is the geometry column name is replaced with the function name. The function's parameters are the longitude and latitude columns from the CITIES_XY table. A geodetic R-tree function-based spatial index is created.

Finally, it is possible to query the table using spatial operators by specifying the function call which returns the SDO_GEOMETRY object. In the example given, all of the cities are returned that are within 100 miles of the given point. This example will use the function-based index for the get_geom function.

Function-Based Indexes Example

```
-- Function as a member function of an object

create or replace type Address_T2 as object
( longitude number,
  latitude number,
  member function GetGeometry(SELF IN Address_T2)
    return mdsys.sdo_geometry DETERMINISTIC);
/

create or replace type body Address_T2 as
  member function GetGeometry(SELF in Address_T2)
  return mdsys.sdo_geometry is
  begin
    return mdsys.sdo_geometry(2001, 8307,
      mdsys.sdo_point_type(longitude,latitude,NULL),
      NULL,NULL);
  end;
end;
/

create table CUSTOMERS2(
  name varchar2(100),
  address Address_T2);
```

ORACLE

11-30

Copyright © Oracle Corporation, 2001. All rights reserved.

Function-Based Indexes Example (continued)

In this example, a function based index is create on the member function of an object.

In the first part of the example, a type is created called Address_T2 which contains a longitude value, a latitude value, and a member function called GetGeometry which takes as input the type Address_T2.

After the type is created, the type body is created. The type body contains the implementation of the GetGeomtry member function. The member function GetGeometry takes as input the Address_T2 type, and returns an MDSYS.SDO_GEOMETRY object. The member function simply returns an MDSYS.SDO_GEOMETRY constructor with the following settings:

- SDO_GTYPE=2001 (always a two-dimensional point)
- SDO_SRID = 8307 (which is WGS 84)
- SDO_POINT constructor populated with the longitude and latitude values passed in
- SDO_ELEM_INFO = NULL
- SDO_ORDINATE_ARRAY = NULL

Next, the CUSTOMERS2 table is created that contains a column of type Address_T2.

Function-Based Indexes Example

```
INSERT INTO user_sdo_geom_metadata values
('CUSTOMERS2',
 'SCOTT.ADDRESS_T2.GETGEOMETRY(ADDRESS)',
 mdsys.sdo_dim_array(
   mdsys.sdo_dim_element('LONG', -180, 180, 0.5),
   mdsys.sdo_dim_element('LAT', -90, 90, 0.5)),
 8307);

CREATE INDEX cust2_sidx on
CUSTOMERS2(address.getgeometry())
INDEXTYPE IS mdsys.spatial_index;

SELECT a.Name
FROM CUSTOMERS2 a
WHERE sdo_within_distance(
  a.address.getgeometry(),
  mdsys.sdo_geometry(2001,8307,
    mdsys.sdo_point_type(-73,42,NULL),NULL,NULL),
  'distance=10 unit=mile') = 'TRUE';
```

ORACLE

11-31

Copyright © Oracle Corporation, 2001. All rights reserved.

Function-Based Indexes Example (continued)

Continuing the previous example, next the metadata is inserted into the USER_SDO_GEOM_METADATA view. The table name is CUSTOMERS2 (this is a table that has the column of type Address_T2. The column name is SCOTT.ADDRESS_T2.GETGEOMETRY(ADDRESS) where:

- SCOTT is the owner of the ADDRESS_T2 type.
- ADDRESS_T2 is the type.
- GETGEOMETRY is the member function.
- ADDRESS is the column of type ADDRESS_T2 passed in as the argument to the member function.

The rest of the USER_SDO_GEOM_METADATA view is filled in as previously discussed. The create index statement is then executed to create the function-based spatial index. In this case a geodetic R-tree index is created on the result of the GETGEOMETRY member function applied to each row of the CUSTOMERS2 table.

After the function-based index is created, the table can be queried using spatial operators, and take advantage of the function-based index. In this example, all of the customer names are returned that are within 10 miles of the given point. Note this query requires a table alias (in this case it is "a"). This is a requirement in Oracle whenever directly accessing attributes or member functions associated with an object.

Privileges for Function-Based Indexes

To create and use function-based indexes the following privileges are required:

- **User privileges:** Users need 'query rewrite' privilege

SQL> grant query rewrite to user

- **Session privileges:** Users need the following session-based privileges

SQL> alter session set QUERY_REWRITE_ENABLED = TRUE;

SQL> alter session set QUERY_REWRITE_INTEGRITY = TRUSTED;

ORACLE

11-32

Copyright © Oracle Corporation, 2001. All rights reserved.

Privileges for Function-Based Indexes

Function-based indexes require specific user and session based privileges.

Every user must have the query rewrite privilege granted. This privilege has to be granted from an account that is able to grant the privilege.

Also the session privilege QUERY_REWRITE_ENABLED must be set to TRUE, and the privilege QUERY_REWRITE_INTEGRITY must be set to TRUSTED. These parameters can be set by default as database initialization parameters. These session-based privileges can be set by any user.

Performance of Function-Based Indexes

- **Create index**
 - The function is called for each row in the table
- **Query performance**
 - **Index only (SDO_FILTER)**
 - No overhead (function call) unless select list includes the geometry
 - **Secondary filter (SDO_RELATE, SDO_NN, and so on)**
 - Filter portion has no overhead
 - Relate will need to instantiate geometries (function call is required)

ORACLE

11-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Performance of Function-Based Indexes

Function-based indexes affect spatial index creation performance. Depending on the type of query and the results desired, function based indexes may also affect query performance.

During the spatial index creation process, the function that returns the MDSYS.SDO_GEOMETRY object is called for every row in the table. The time it takes to build the index includes the overhead associated with calling the function for each row. In the function-based index examples previously discussed, the functions were very simple. They simply returned a geometry constructor. If the function is more complicated and takes longer to run, the function call overhead of the create index process will also increase..

Any query that performs a SELECT which specifies the function that returns the geometry object will have to instantiate the geometry by calling the function.

At query time, primary filter operators (i.e. SDO_FILTER or SDO_WITHIN_DISTANCE specifying QUERYTYPE=FILTER) do not incur any performance overhead with spatial function-based indexes. Only the spatial-function based index is searched. The function is never called..

At query time, SDO_RELATE, SDO_NN, and SDO_WITHIN_DISTANCE (without QUERYTYPE=FILTER) incur the function call overhead for each candidate returned by the index search that requires additional processing (any geometry that wasn't optimized directly into result set).

Embedded SDO_GEOMETRY Objects

ORACLE

11-34

Copyright © Oracle Corporation, 2001. All rights reserved.

Embedded Spatial Geometry

- **Spatial geometry type can be embedded in another user-defined type**
- **This is useful for building applications requiring spatial and nonspatial attributes in a type**
- **Examples**
 - **GeoImage: Image and Geometry**
 - **MapType: Drawing attributes and shape**

ORACLE

11-35

Copyright © Oracle Corporation, 2001. All rights reserved.

Embedded Spatial Geometry

Before Oracle9i, Oracle Spatial indexes could not be built of the MDSYS.SDO_GEOMETRY type was embedded in another type. With the advent of Oracle9i, this restriction has been lifted, and objects of type MDSYS.SDO_GEOMETRY can be embedded in other user-defined types and spatial indexes can be created and used. This is extremely useful for applications that require both spatial and nonspatial attributes encapsulated in a single type.

Some sample applications include:

- Geoimagery applications, where a type is defined to contain the perimeter of an image and the image itself. The type could be defined as containing an MDSYS.SDO_GEOMETRY field for the perimeter, and a BLOB field to store the image.
- Mapping applications, where a Map type includes drawing attributes (i.e. color, line style etc.), and the associated geometry.

Embedded Geometry Procedure

- **Create the new user-defined type**
- **Create and populate a column of that type**
- **Insert the geometry metadata**
- **Create spatial index on the geometry attribute**
- **Perform spatial queries**

ORACLE

11-36

Copyright © Oracle Corporation, 2001. All rights reserved.

Embedded Geometry Procedure

The procedure to index and query embedded types is similar to the procedure for nonembedded SDO_GEOMETRY columns. There are some minor syntax changes:

- The user-defined type is created first.
- Then a table is created that has a column of that type. The column is populated with appropriate data.
- The USER_SDO_GEOM_METADATA view is populated.
- An index can then be created on the MDSYS.SDO_GEOMETRY attribute of the user-defined type.
- Spatial operators use the embedded attribute's spatial index.

Embedded Geometry Example

```
create or replace type Street_Address as Object (  
    Street varchar2(200),  
    City   varchar2(200),  
    State  char(2),  
    Zip    varchar2(20));  
/  
create or replace type Address_T as object (  
    St_Address Street_Address,  
    Location   mdsys.sdo_geometry);  
/  
create table customers(  
    name      varchar2(100),  
    address   Address_T);
```

ORACLE

11-37

Copyright © Oracle Corporation, 2001. All rights reserved.

Embedded Geometry Example

In this example, the following two user-defined types are created:

- Street_Address: Contains Street, City, State, and Zip attributes (or fields)
- Address_T: Contains St_Address of Street_Address type, and Location of type MDSYS.SDO_GEOMETRY

A table is created with two columns, NAME and ADDRESS. Note that ADDRESS is of type Address_T, and contains the information described above.

Embedded Geometry Example

```
insert into user_sdo_geom_metadata values (
  'CUSTOMERS',
  'ADDRESS.LOCATION',
  mdsys.sdo_dim_array (
    mdsys.sdo_dim_element ('Long', -180, 180, 0.5),
    mdsys.sdo_dim_element ('Lat', -90, 90, 0.5)), 8307);

insert into customers values (...)...

create index cust_sidx on customers(address.location)
indextype is mdsys.spatial_index;

select a.Name
from customers a
where sdo_within_distance(
  a.address.location,
  mdsys.sdo_geometry(2001, 8307,
    mdsys.sdo_point_type(-71.45937,42.70782,NULL),NULL,NULL),
  'distance = 5 unit = MILE') = 'TRUE';
```

ORACLE

11-38

Copyright © Oracle Corporation, 2001. All rights reserved.

Embedded Geometry Example (continued)

Continuing the example, the USER_SDO_GEOM_METADATA view is populated with the following:

- CUSTOMERS is the table name.
- The column name is the embedded geometry attribute in the type. In this example, it is the LOCATION attribute of the ADDRESS column.
- The rest of the information inserted into the USER_SDO_GEOM_METADATA view is the same as has been described previously.

Next, data is inserted into the table, and a spatial index is created. In this example, an R-tree index is created called CUST_SIDX on the LOCATION attribute of the ADDRESS column in the CUSTOMERS table.

Finally, a spatial query is shown that returns the name of any customer within 5 miles of (-71.45937, 42.70782). Note that this query requires a table alias (in this case it is “a”). This is a requirement in Oracle whenever accessing attributes or member functions of an object.

Summary

In this lesson, you should have learned how to:

- **Create and use partitioned spatial indexes**
- **Create and use function-based indexes**
- **Embed and index spatial objects within objects**

ORACLE

11-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Summary

This lesson described the process of building spatial indexes using advanced indexing mechanisms.

12

Linear Referencing System (LRS)

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Understand LRS concepts**
- **Understand the functions available in LRS**
- **Understand how to implement linear referencing in Oracle Spatial**

ORACLE

What Is LRS?

- **A technique to associate events or attributes to locations or portions of a linear feature**
- **Used in transportation systems (roads, railroads, and so on) and utilities (oil and gas pipelines)**
- **Events or attributes located by a *measure* along the linear feature**
- **Locate sections along linear features dynamically without duplicating the referenced geometry (dynamic segmentation)**

ORACLE

12-3

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS (Linear Referencing System): What Is It?

LRS is a way to store a line string or polygon geometry once and associate many attributes to the geometry, without duplicating the geometry, or even pieces of the geometry.

LRS is commonly used by transportation departments (to model roads or railroads and their associated attributes) or utilities (to model oil or gas pipes and their associated attributes) .

A common usage is to assign mileposts along a highway and call them measure values. These mile posts or measure values are linear. The measure values are proportional to how far you traverse the highway.

In another table (not the table that contains the highway geometries), you can associate measure ranges to nonlinear attributes like speed limit or road condition. Speed limit is not a linear value, it is a discrete value. It does not increase the further you traverse a highway as mileposts do.

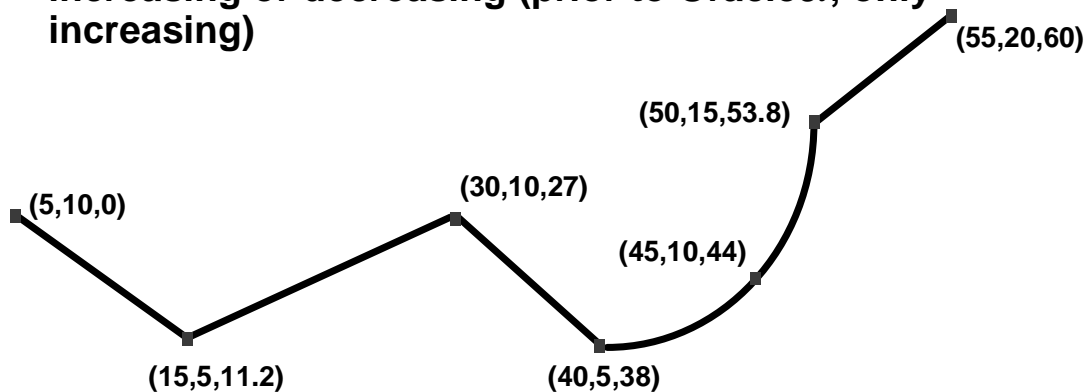
The attribute tables do not duplicate any part of the feature geometry. The attribute table stores measure values (that is, the start and end measure values for a speed limit associated with a highway segment) and the associated attribute values (the speed limit).

LRS (Linear Referencing System): What Is It? (continued)

With the LRS functions in Oracle Spatial, you can perform dynamic segmentation with SQL. Dynamic segmentation dynamically locates events along a geometry (possibly for display). For example, dynamically locate all the segments of highway I5 that have poor pavement condition for display. The dynamic segments you extract do not have to start or end on physical points stored in the original geometry. As you will see later in this lesson, this can be done in a single SQL statement that utilizes Oracle Spatial LRS functions.

What Is 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
- In Oracle9i, measure values can be monotonically increasing or decreasing (prior to Oracle9i, only increasing)



ORACLE

12-5

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is 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.

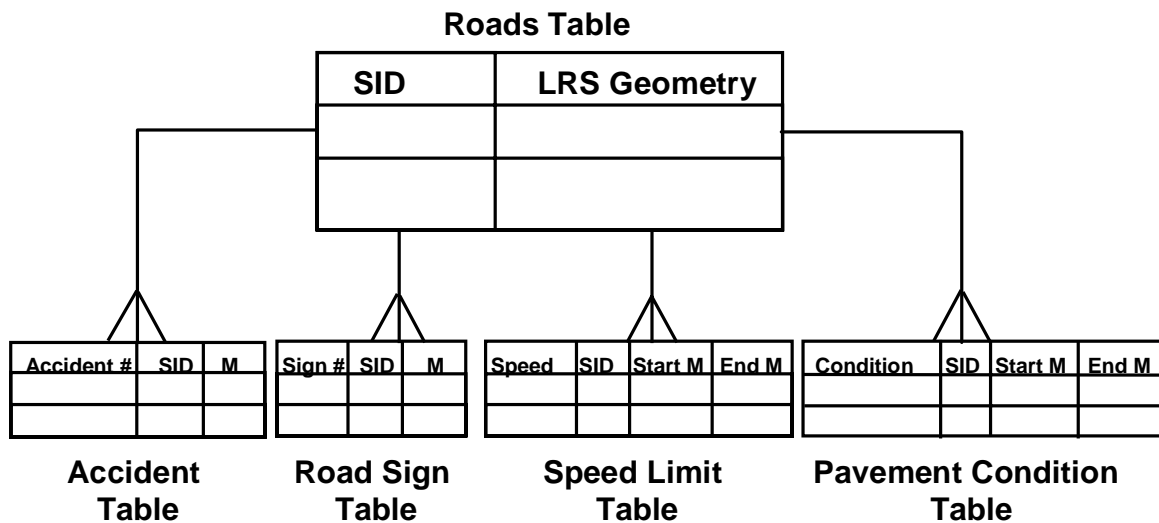
Linear-referenced geometries are stored in one table, and in another table are associated measure value ranges with discrete attributes like speed limits. For example, in the road depicted in this slide, measure values 0 through 27 could represent a 50 mile per hour zone, and measure values 27 through 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 (for example, 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 measures values, taking into account any measure values you have set in the geometry.

LRS Application

Transportation Application in the DBMS



ORACLE

12-6

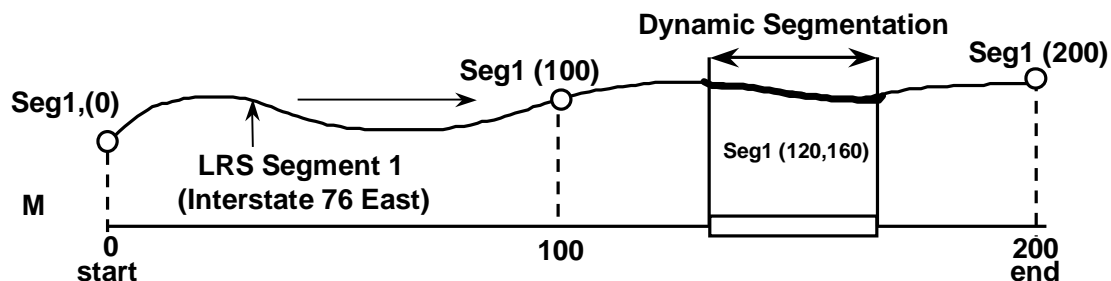
Copyright © Oracle Corporation, 2001. All rights reserved.

Transportation Application in the DBMS

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

LRS Segment

- A closer look at an LRS segment
 - Directional
 - Registered with measure information
 - Line string and multiline string geometries supported (nonbranching)
 - Single polygons with zero or more holes are supported
 - Measure typically proportional to distance



ORACLE

12-7

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Segment

This example is a closer look at a linear-referenced geometry.

Measure values are directional, and can either increase throughout a feature or decrease throughout a feature (but not both increase and decrease).

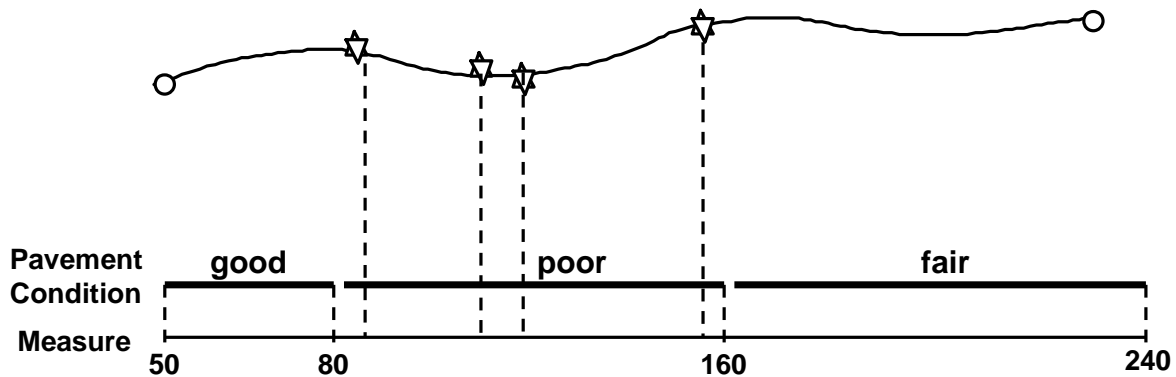
Single and multiline string geometries can be linear referenced (but they cannot branch). Single polygons with zero or more voids can also be linear referenced.

Measure values are typically (but not always) proportional to a distance from the geometry's start point. Control point measures can be set within a feature to register known measure values. An example of setting up measure control points will be shown in this lesson. Control points are not necessarily proportional to distance.

Dynamic segmentation does not have to occur at nodes (stored vertices) in the geometry, it is completely dynamic with regards to the locations where dynamic segmentation can occur.

Associating Events with LRS

- **How to associate events with linear referencing?**
 - **Point Event: Accidents, Stop Signs**
 - **Linear Event: Pavement Conditions**



ORACLE

12-8

Copyright © Oracle Corporation, 2001. All rights reserved.

Associating Events with LRS

You can associated many attributes with the same linear-referenced geometry. For example, the linear feature in this slide has three associated attribute tables (pavement conditions, accidents, and stop signs).

The linear-referenced geometry, along with a unique identifier, are stored in a separate table from the attribute tables.

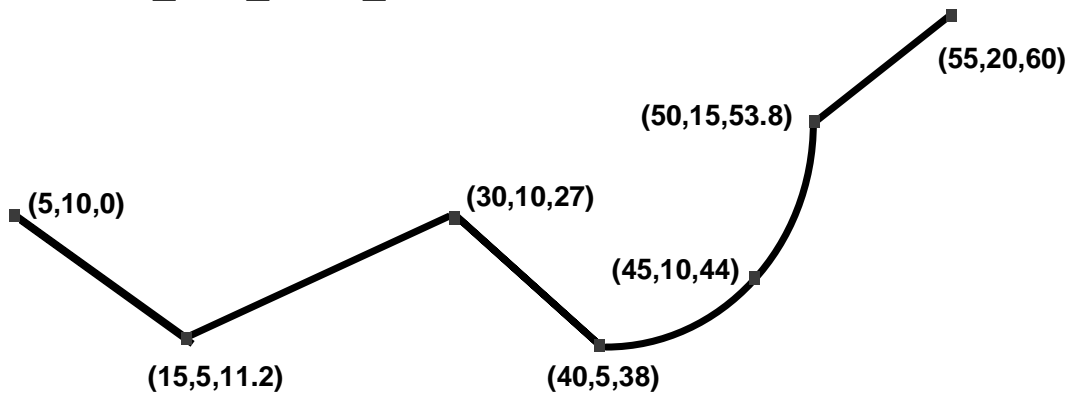
The pavement conditions attribute table has a foreign key back to the table with the geometry, and also a start and end measure to associate a pavement condition with a section of the geometry.

The accidents attribute table has a foreign key back to the table with the geometry, and also a measure value to associate an accident with a point on the geometry.

The stop signs attribute table has a foreign key back to the table with the geometry, and also a measure value to associate a stop sign with a point on the geometry.

Defining an LRS Geometry

- Use one more ordinate in each point of a geometry to define the measure of that point.
- Define one more dimension in the `USER_SDO_GEOM_METADATA` view for the measure.



ORACLE

12-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Defining an LRS Geometry

This is an example of a 2D line string with a measure value associated with each point (that is, (x1, y1, measure1, x2, y2, measure2, and so on)).

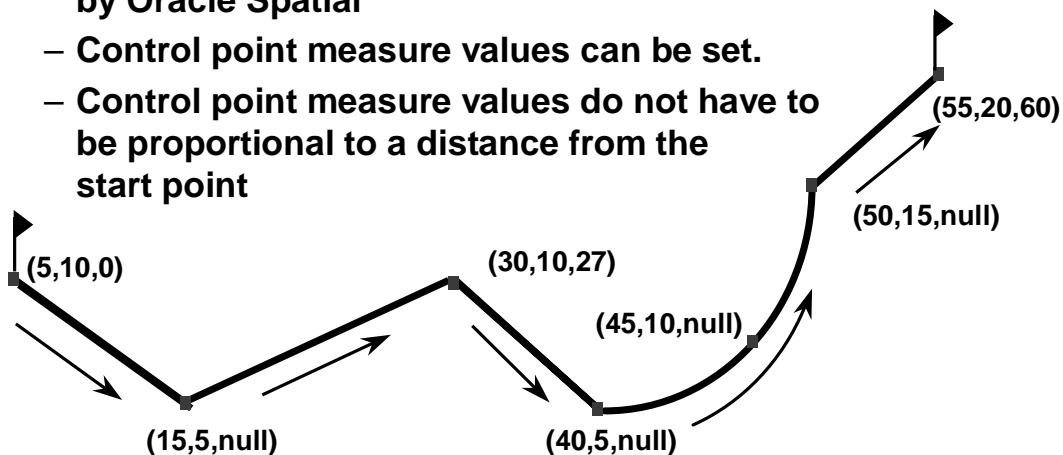
In this example, a measure value is provided for each point. The measure values range from 0 to 60.

The `USER_SDO_GEOM_METADATA` view requires a dimension defined in the `diminfo` array for the measure value. The lower and upper bound you enter for the measure is not currently used by Oracle Spatial (that is, no integrity checks are made against the measure upper and lower bound). The tolerance value is important, and is used by some LRS functions, for example `SDO_LRS.CONNECTED_GEOM_SEGMENTS` which will be discussed later.

LRS Concepts

- **Shape points**

- Points along the geometry that have associated measures
- Correspond to geometry vertices
- NULL shape point measure values can be computed by Oracle Spatial
- Control point measure values can be set.
- Control point measure values do not have to be proportional to a distance from the start point



ORACLE

12-10

Copyright © Oracle Corporation, 2001. All rights reserved.

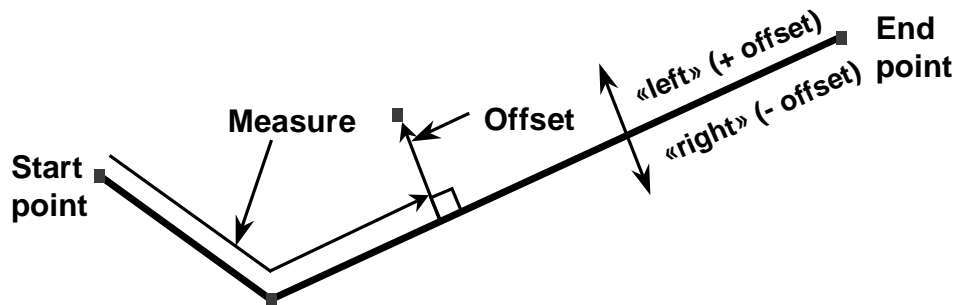
LRS Concepts

Shape points

- Points along a geometry that have associated measure values.
- Shape points correspond to geometry vertices
- Shape points can have null measure values. The `SDO_LRS.DEFINE_GEOM_SEGMENT` (discussed in an upcoming slide) can interpolate the geometry and set all the NULL measure values.
- Typically, a point's measure value is proportional to its distance from the geometry's start point. But this is not always the case. For example, a 2D line string whose start point and measure value is (0,0,0) and whose end point is (10,10,10), can have a control point set to (5,5,8). Even though (5,5) physically corresponds to the line string midpoint, the measure value interpolation will account for the steep slope between (0,0) and (5,5).
- As will be seen in upcoming slides:
 - If data is projected, Oracle Spatial will take into account slopes (for example, Z) when it computes measure values for shape points.
 - If data is geodetic, Oracle spatial does not consider Z when it computes measure values for shape points. Setting control points as described above can help you account for Z when your data is geodetic.
- Measure values are either all increasing within a geometry, or all decreasing, but not both.

LRS Concepts

- **Direction of the segment**
 - Defined by the order the shape points are listed
- **Measure of a point**
 - Measure associated with a point along the geometry
- **Offset of a point**
 - Perpendicular distance from a segment of the geometry feature to a point. Positive offsets are to the left, negative offsets to the right.



12-11

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE

LRS Concepts (continued)

Direction of a Segment

The direction is defined by the order in which the shape points are listed in the SDO_ORDINATES array.

Measure of a Point

The linear distance (that is, measure distance) from the start point of a geometry to a point on that geometry.

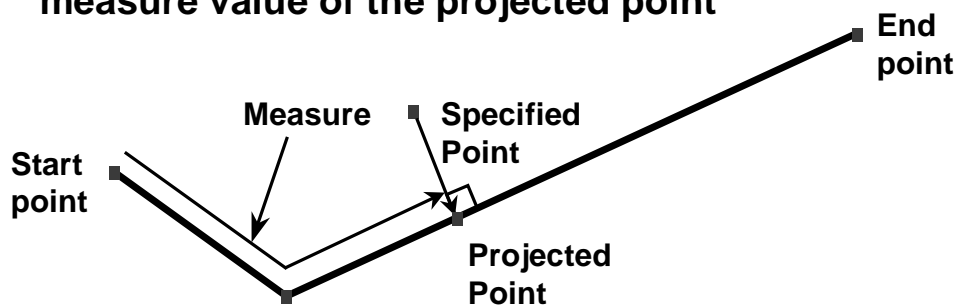
Offset of a Point

The perpendicular distance from a geometry segment to a point. Offsets can be positive or negative. For example, offsets are commonly used to specify the location of a stop sign, or a guard rail, offset from a road.

LRS Concepts

Projection of a point on a segment

- Projects a specified point onto a linear-referenced geometry (that is, the point on the segment with the minimum distance to the specified point)
- The projection operation returns the point and measure value of the projected point



ORACLE

12-12

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Concepts (continued)

Projection of a Point

Projects a specified point onto a linear-referenced geometry (that is, the closest point on the linear-referenced geometry to the specified point). As will be discussed later in the lesson, the projection operation returns an `SDO_GEOMETRY` object with a point and measure value. The `SDO_GEOMETRY` object returned will fall on the linear-referenced geometry.

SDO_GTYPE with LRS: Oracle9i and Beyond

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

12-13

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 (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 (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 (SDO_GTYPE 4401, 4402, 4403, or 4406).

SDO_GTYPE with LRS: Oracle9i and Beyond (continued)

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

12-15

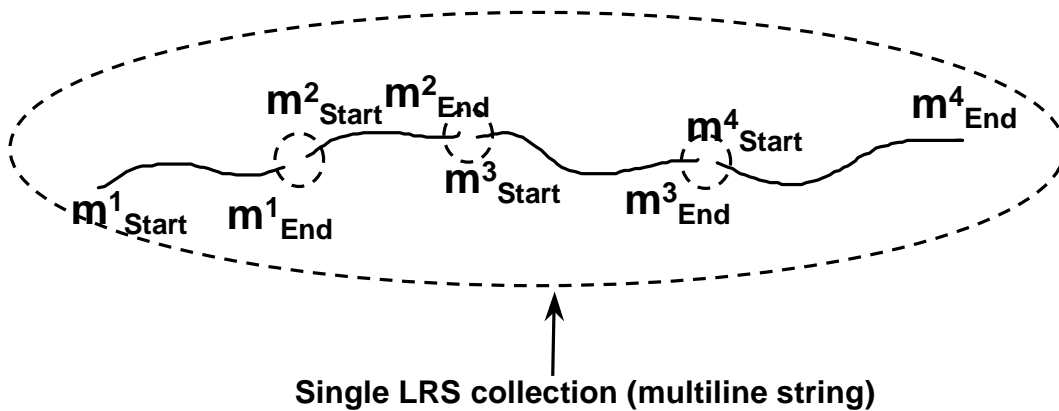
Copyright © Oracle Corporation, 2001. All rights reserved.

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.

LRS Support for Collections

- Supports collection of LRS geometries (multiline strings only)
- Line segments in a collection must be nonbranching



ORACLE

12-16

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Support for Collections

In Oracle9i, multiline string geometries can be linear referenced. Only nonbranching geometries can be linear referenced.

LRS Support for Collections

LRS collection

- **SDO_GTYPE is dm06 (multiline string)**
 - **d represents dimensionality**
 - **m represents the measure position**
 - **6 indicates this is a multiline string**
- **Line segment order is used to assign measure values**
- **Disjoint line segments can have same or different measure values at the joint**

ORACLE

12-17

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Support for Collections (continued)

The SDO_GTYPE for a linear-referenced multiline string is dm06 where:

- d represents the dimensionality (how many numbers make up each point)
- m represents the measure position
- 6 indicates this geometry is a multiline string

If Oracle Spatial does the measure interpolation for a multiline string, the line string segments will be interpolated in the order they appear in the geometry.

Multiline string joints (that is, where gaps appear in the multiline string) can either have the same measure value, or different measure values. If Oracle Spatial does the interpolation, the same measure value will be assigned at multiline string joints. If you want different measure values at the joints, you can assign control point measure values.

Defining LRS Structures

- Use the standard `MDSYS.SDO_GEOMETRY` object
- Define an extra dimension in the metadata
 - 3rd or 4th dimension for a 2D geometry
 - 4th dimension for a 3D geometry

```
SQL> INSERT INTO user_sdo_geom_metadata VALUES
2>   ('LRS_HIGHWAYS',
3>    'GEOMETRY',
4>    MDSYS.SDO_DIM_ARRAY (
5>      MDSYS.SDO_DIM_ELEMENT('Long', -180, 180, 0.5),
6>      MDSYS.SDO_DIM_ELEMENT('Lat', -90, 90, 0.5),
7>      MDSYS.SDO_DIM_ELEMENT('Measure', 0, 1000, 0.5)
8>    ), 8307);
```

ORACLE

12-18

Copyright © Oracle Corporation, 2001. All rights reserved.

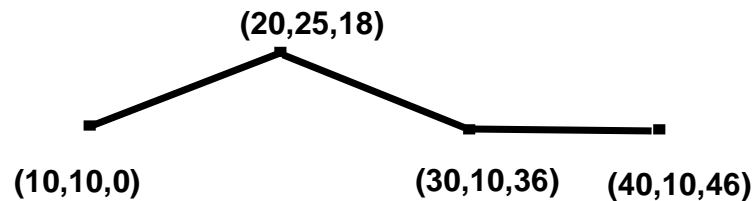
Defining LRS Structures

The `USER_SDO_GEOM_METADATA` view requires you to define a dimension (in `diminfo`) for the measure value. The lower and upper bound you enter for the measure is not currently used by Oracle Spatial (that is, no integrity checks are made against the measure upper and lower bound). The tolerance value is important, and is used by some LRS functions, for example `SDO_LRS.CONNECTED_GEOM_SEGMENTS` discussed in an upcoming slide.

- 2D geometry with a measure
 - 2D geometries with a measure value must have the measure value as the third dimension.
- 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 dimension
- 3D geometry with a measure
 - 3D geometries (x,y,z values) and a measure must store the measure value in the fourth dimension.

The example in this slide is a 2D geometry with a measure stored in the third position.

Constructing Geometries with LRS: All Measures Are Known



```
SQL> INSERT INTO lines VALUES (  
2>     attribute_1, ... attribute_n,  
3>     mdsys.sdo_geometry (  
4>         3302, null, null,  
5>         mdsys.sdo_elem_info_array (1,2,1),  
6>         mdsys.sdo_ordinate_array (  
7>             10,10,0, 20,25,18, 30,10,36, 40,10,46))  
8>     );
```

ORACLE

12-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Constructing Geometries with LRS: All Measures Are Known

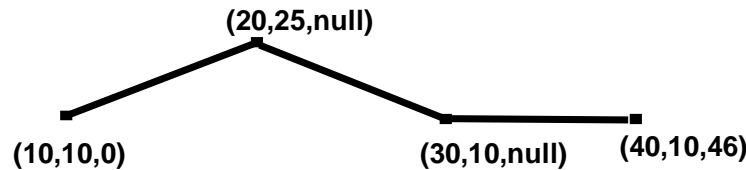
Constructing geometries with LRS is similar to constructing geometries without LRS.

The SDO_GTYPE must contain the LRS position (in the second digit from the left).

A measure value is included for each point in the SDO_ORDINATES array. The measure value must be in the correct position (denoted in the SDO_GTYPE) for each point.

In this example, all measure values are known. The measure is placed in the third position of each point of the SDO_ORDINATES array.

Constructing Geometries with LRS: Some Measures Are Unknown



```
SQL> INSERT INTO lines VALUES (  
2>     attribute_1, ... attribute_n,  
3>     mdsys.sdo_geometry (  
4>         3302, null, null,  
5>         mdsys.sdo_elem_info_array (1,2,1),  
6>         mdsys.sdo_ordinate_array (  
7>             10,10,0, 20,25,null, 30,10,null, 40,10,46))  
8> );
```

- Use the `SDO_LRS.DEFINE_GEOM_SEGMENT` function to populate missing measures by interpolation.

ORACLE

12-20

Copyright © Oracle Corporation, 2001. All rights reserved.

Constructing Geometries with LRS: Some Measures Are Unknown

When building LRS geometries, there may be shape points that do not have measure values (that is, measure values are set to null). Oracle Spatial can interpolate the distance between measure values that are not null, and fill in measures for shape points that are null.

If the start and end measures are omitted in a call to `SDO_LRS.DEFINE_GEOM_SEGMENT`, the start measure will default to 0, and the length of the geometry is used for the end measure. Oracle Spatial will interpolate all the measure values for intermediate shape points.

If you want to set the measure value for some of the shape points, and let Oracle Spatial interpolate only the shape points with NULL measure values, you must pass start and end measure values to `SDO_LRS.DEFINE_GEOM_SEGMENT`.

Null measure values will be set to the proportional distance along the line segment defined by the previous and next points with non-null measure values.

`SDO_LRS.DEFINE_GEOM_SEGMENT` does not change the stored geometries, unless the function is invoked with an INSERT or UPDATE statement.

SDO_LRS.CONVERT_TO_LRS_GEOM

- **Single non-LRS geometry conversion**
 - Convert a non-LRS 2D geometry into an LRS geometry
 - 2D geometry can have a non-Z third dimension
 - Measures calculated by using first two dimensions only
 - In-memory conversion (changed geometry not stored)

```
lrs_geom := sdo_lrs.convert_to_lrs_geom(  
                                std_geom,  
                                measure_pos, -- Either 3 or 4  
                                start_measure, end_measure);
```

LRS in Oracle Spatial: SDO_LRS.CONVERT_TO_LRS_GEOM

To convert a single geometry with no measures to an LRS geometry (that is, with measures), you can use `SDO_LRS.CONVERT_TO_LRS_GEOM`.

This function will return a new `SDO_GEOMETRY` object with an extra ordinate for each point in the geometry. The extra ordinate will be the measure value at that point.

The non-LRS geometry passed in can either be:

- 2D geometry
 - Output LRS geometry will be a 2D geometries with a measure value in the third position of each point
- 2D geometry with and another non-Z value dimension
 - Output LRS geometry will be a 2D geometries with a non-Z value and a measure. You can specify the measure to be in position 3 or 4 for each point that makes up the LRS geometry.

In both input geometry scenarios above, measure values are proportional to a point's distance from the start point of the geometry. Distances are always calculated by using only the first two dimensions of each point. If the geometry is geodetic, and start and end measures are omitted, geodetic distances (takes into account the Earth's curvature) are used to set measure values. The geodetic distance unit for the measure will be meters.

SDO_LRS.CONVERT_TO_LRS_GEOM

Note:

- To take advantage of 3D interpolation for measures, use `CONVERT_TO_LRS_GEOM_3D`. The `_3D` functions take a Z value into account when interpolating measures, but cannot be used for geodetic data.
- For 2D geodetic segments, if you do not specify the start and end measures, a geodetic length (in meters) will be used for the measure interpolation.

ORACLE

12-22

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS in Oracle Spatial: SDO_LRS.CONVERT_TO_LRS_GEOM (continued)

To take advantage of 3D distances between points (that is, account for line slope in distance calculation for measures), use the 3D version of this routine (`CONVERT_TO_LRS_GEOM_3D`). 3D functions cannot be used on geodetic data.

SDO_LRS.CONVERT_TO_LRS_LAYER

- **Converts entire layer of non-LRS geometries:**

```
result := sdo_lrs.convert_to_lrs_layer(  
    std_geom_table, geom_col_name,  
    metadata_lower_bound,  
    metadata_upper_bound,  
    metadata_tolerance);
```

- **Must drop spatial index (if it exists) prior to calling this routine**
- **CONVERT_TO_LRS_LAYER_3D version available**
- **Start measure is always 0**

Note: Updates the USER_SDO_GEOM_METADATA view with the measure bounds information and tolerance.

ORACLE

12-23

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS in Oracle Spatial: SDO_LRS.CONVERT_TO_LRS_LAYER

To convert an entire layer of geometries with no measures to a layer of LRS geometries (for example, with measures), you can use SDO_LRS.CONVERT_TO_LRS_LAYER.

For each geometry in the layer, this function updates the geometry with an LRS geometry. The geometries in the layer can be:

- 2D geometry
 - Updated LRS geometry will be a 2D geometries with a measure value in the third position of each point
- 3D geometry
 - Updated LRS geometry will be 3D geometries with a measure value in the fourth position of each point (this function will not take into account the third dimension when calculating measures).

In both input geometry scenarios above, measure values are proportional to a point's distance from the start point of the geometry. Distances are always calculated by using only the first 2 dimensions of each point. If the geometry is geodetic then geodetic distances (which take into account the Earth's curvature) are used to set measure values. The geodetic distance unit for the measure will be meters.

To take advantage of 3D distances between points (for example, account for line slope in distance calculation for measures), use the 3D version of this routine (CONVERT_TO_LRS_GEOM_3D). 3D functions cannot be used on geodetic data.

Overview of LRS Functions: Geometry Manipulation

- `SDO_LRS.DEFINE_GEOM_SEGMENT (geom, mS, mE)`
 - Defines a geometric segment, fills in NULL measure values
- `SDO_LRS.CLIP_GEOM_SEGMENT (geom, m1, m2)`
 - Clips a geometric segment
- `SDO_LRS.SPLIT_GEOM_SEGMENT (geom, m, segment_1, segment_2)`
 - Splits a geometric segment into two (procedure)
- `SDO_LRS.CONCATENATE_GEOM_SEGMENTS (geom1, geom2)`
 - Concatenates two geometric segments into one

ORACLE

12-24

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Geometry Manipulation

- `SDO_LRS.DEFINE_GEOM_SEGMENT (geom, mS, mE)`
 - Fills in all the NULL measure values of an LRS geometry.
- `SDO_LRS.CLIP_GEOM_SEGMENT (geom, m1, m2)`
 - Given a start and end measure, returns the clipped segment.
- `SDO_LRS.SPLIT_GEOM_SEGMENT (geom, m, segment_1, segment_2)`
 - Given a measure value, splits an LRS geometry into two segments. This is a procedure with `segment_1` and `segment_2` as output parameters.
- `SDO_LRS.CONCATENATE_GEOM_SEGMENTS (geom1, geom2)`
 - Given two LRS geometries, returns a single LRS geometry that is the concatenation of the two input geometries.

Overview of LRS Functions: Geometry Manipulation

- `SDO_LRS_AGGR_CONCAT (SDOAGGRTYPE(geom, tolerance))`
 - Returns aggregate concatenation of LRS geometries passed in
- `SDO_LRS.TRANSLATE_MEASURE (geom, m)`
 - Translates the measure values by a specified amount
- `SDO_LRS.OFFSET_GEOM_SEGMENT (geom, start_m, end_m, offset)`
 - Returns a clip of an LRS geometry, offset from the original LRS geometry by the specified amount

ORACLE

12-25

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Geometry Manipulation (continued)

- `SDO_LRS_AGGR_CONCAT(SDOAGGRTYPE(geom, tolerance))`
 - Given any number of LRS geometries, performs an aggregate concatenation. Returns a single LRS geometry that concatenates the specified LRS geometries.
- `SDO_LRS. TRANSLATE_MEASURE (geom, m)`
 - Translates each measure value of an LRS geometry by a specified amount.
- `SDO_LRS.OFFSET_GEOM_SEGMENT (geom, start_m, end_m, offset)`
 - Given a start measure, end measure, and an offset, returns a clipped LRS geometry at the specified offset from the original LRS geometry.

Overview of LRS Functions: Geometry Manipulation

- `SDO_LRS.REVERSE_GEOMETRY (geom)`
 - Returns an LRS geometry that reverses the points and the measure information
- `SDO_LRS.REVERSE_MEASURE (geom)`
 - Returns an LRS geometry that reverses the measure information, but not the points that make up the geometry
- and others...

ORACLE

12-26

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Geometry Manipulation (continued)

- `SDO_LRS.REVERSE_GEOMETRY (geom)`
 - Reverses the measure values of an LRS geometry. Returns a new LRS geometry by reversing the measure values and the direction (order of vertices) of the original LRS geometry.
- `SDO_LRS.REVERSE_MEASURE (geom)`
 - Reverses the measure values of an LRS geometry. Returns a new LRS geometry by reversing the measure values, but not the direction (order of vertices) of the original LRS geometry.
- and others...

Overview of LRS Functions: Geometry Interrogation

- `SDO_LRS.IS_MEASURE_INCREASING (geom)`
 - Returns **TRUE** if the measure values are increasing
- `SDO_LRS.IS_MEASURE_DECREASING (geom)`
 - Returns **TRUE** if the measure values are decreasing
- and others...

ORACLE

12-27

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Geometry Interrogation

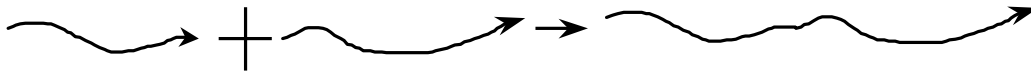
- `SDO_LRS.IS_MEASURE_INCREASING (geom)`
 - Checks if the measure values along an LRS geometry are increasing (ascending in numerical value). Returns **TRUE** if the measure values are increasing and **FALSE** if they are not increasing.
- `SDO_LRS.IS_MEASURE_DECREASING (geom)`
 - Checks if the measure values along an LRS geometry are decreasing (descending in numerical value). Returns **TRUE** if the measure values are decreasing and **FALSE** if they are not decreasing.
- and others ...

LRS In Oracle Spatial: Linear Functions

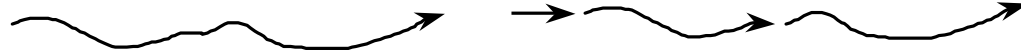
Clipping (DynSeg)



Concatenation



Splitting



Offset



ORACLE

12-28

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS in Oracle Spatial

This slide demonstrates the results of some of the LRS functions previously described. Clipping dynamically returns a piece of a linear feature based on a start and end measure. Concatenation takes as input two linear features and concatenates them. Splitting takes a single linear feature and splits it into two based on a measure value. Offset returns a clipped geometry offset by a specified amount.

Overview of LRS Functions: Point Functions

- All LRS functions that return a point populate `SDO_ELEM_INFO` and `SDO_ORDINATES` (not the `SDO_POINT` field)
- `SDO_LRS.GET_MEASURE (point)`
 - Returns the measure of an LRS point (that is, the measure value)
- `SDO_LRS.LOCATE_PT (geom, measure, offset)`
 - Finds the location of a point described by a measure and an offset on a geometric segment

ORACLE

12-29

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Functions: Point Functions

Any LRS function that returns a point geometry, will return the point in the `SDO_ELEM_INFO` and `SDO_ORDINATES` fields (not the `SDO_POINT` field).

- `SDO_LRS.GET_MEASURE (point)`
 - Returns the measure value of an LRS point (that is, the measure value).
- `SDO_LRS.LOCATE_PT (geom, measure, offset)`
 - Finds the location of a point described by a measure and an offset on a geometric segment.

Overview of LRS Functions: Point Functions

- `SDO_LRS.PROJECT_PT (geom, in_point)`
 - Returns the projection point
 - Projection point returned is the point on the LRS geometry closest to `<in_point>`
 - The projected point returned will contain its associated measure value.

ORACLE

12-30

Copyright © Oracle Corporation, 2001. All rights reserved.

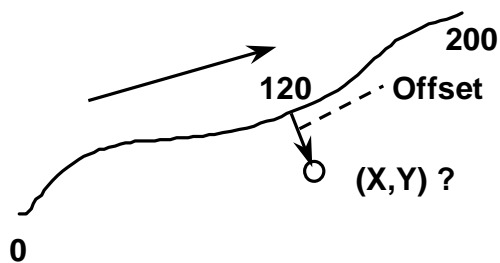
LRS Functions: Point Functions (continued)

- `SDO_LRS.PROJECT_PT (geom, in_point)`
 - Returns the projection point.
 - Projection point returned is the point on the LRS geometry closest to `<in_point>`.
 - The projected point returned will contain its associated measure value.

LRS in Oracle Spatial: Point Functions

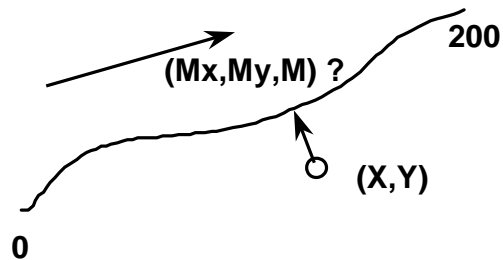
Locate Point

(LRS Geom, M, Offset) -> (X,Y)



Project Point

(LRS Geom, X,Y) -> (Mx,My,M)



ORACLE

12-31

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS in Oracle Spatial: Point Functions

Locate Point

Given a measure and an offset, traverse the LRS geometry and locate the desired point. If the offset is 0, the point will lie on the LRS geometry.

Project Point

Given a point (x,y) that lies off the LRS segment, find the closest point on the LRS geometry to the given point (Mx, My, M). The projected point returned contains a measure value.

Overview of LRS Functions: Inspector Functions

- `SDO_LRS.GEOM_SEGMENT_LENGTH (geom)`
 - Returns the length of a geometric segment (not measure length).
- `SDO_LRS.GEOM_SEGMENT_START_PT (geom)`
 - Returns the start point of a geometric segment
 - Returned point includes measure value
- `SDO_LRS.GEOM_SEGMENT_END_PT (geom)`
 - Returns the end point of a geometric segment
 - Returned point includes measure value

ORACLE

12-32

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of LRS Functions: Inspector Functions

- `SDO_LRS.GEOM_SEGMENT_LENGTH (geom)`
 - Returns the length of a geometric segment (not measure length).
- `SDO_LRS.GEOM_SEGMENT_START_PT (geom)`
 - Returns the start point of a geometric segment.
 - Returned point includes measure value.
- `SDO_LRS.GEOM_SEGMENT_END_PT (geom)`
 - Returns the end point of a geometric segment.
 - Returned point includes measure value.

Overview of LRS Functions: Inspector Functions

- `SDO_LRS.GEOM_SEGMENT_START_MEASURE (geom)`
 - Returns the start measure of a geometric segment
- `SDO_LRS.GEOM_SEGMENT_END_MEASURE (geom)`
 - Returns the end measure of the a geometric segment

ORACLE

12-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of LRS Functions: Inspector Functions (continued)

- `SDO_LRS.GEOM_SEGMENT_START_MEASURE (geom)`
 - Returns the start measure of a geometric segment.
- `SDO_LRS.GEOM_SEGMENT_END_MEASURE (geom)`
 - Returns the end measure of the a geometric segment.

Overview of LRS Functions: Validation Functions

These functions are used in addition to `VALIDATE_GEOMETRY` or `VALIDATE_LAYER`.

- `SDO_LRS.VALID_GEOM_SEGMENT (geom)`
 - Only checks geometry type and the number of dimensions for each point is accurate
 - Use with `IS_GEOM_SEGMENT_DEFINED`
- `SDO_LRS.IS_GEOM_SEGMENT_DEFINED (geom)`
 - Checks for non-null start and end measures, and all non-null measure values are either ascending or descending

ORACLE

12-34

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of LRS Functions: Validation Functions

- `SDO_LRS.VALID_GEOM_SEGMENT (geom)`
 - Only checks geometry type and the number of dimensions for each point is accurate.
 - Use with `IS_GEOM_SEGMENT_DEFINED`.
- `SDO_LRS.IS_GEOM_SEGMENT_DEFINED (geom)`
 - Checks for non-null start and end measures, and all non-null measure values are either ascending or descending.

Overview of LRS Functions: Validation Functions

- `SDO_LRS.VALID_LRS_PT (point)`
 - Checks if a LRS point is valid (that is, the number of dimensions in point is correct)
- `SDO_LRS.VALID_MEASURE (geom, measure)`
 - Checks if a measure falls within the measure range of a geometric segment
- `SDO_LRS.CONNECTED_GEOM_SEGMENTS (geom1, geom2)`
 - Checks if two geometric segments are connected

ORACLE

12-35

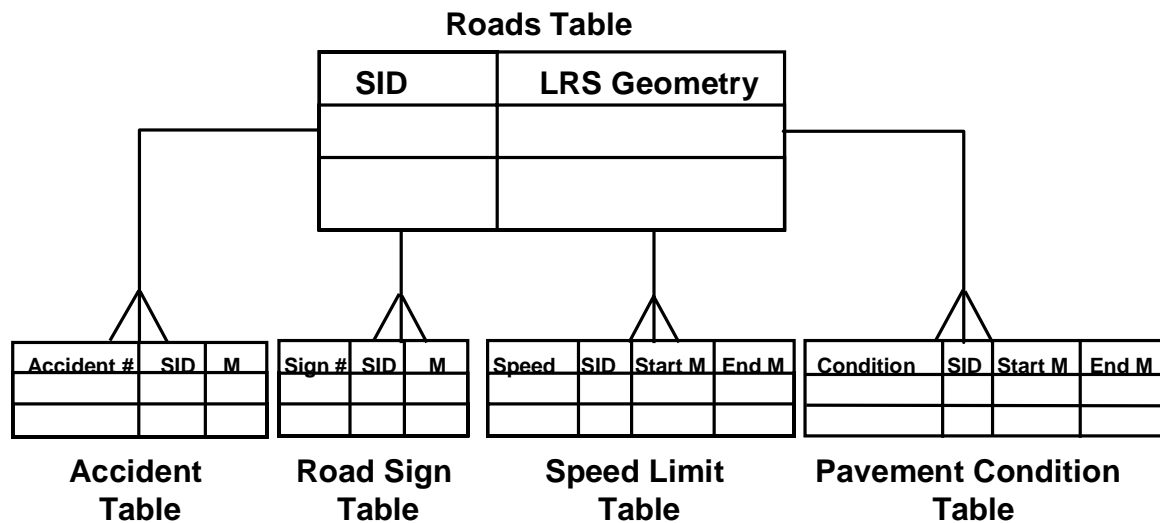
Copyright © Oracle Corporation, 2001. All rights reserved.

Overview of LRS Functions: Validation Functions (continued)

- `SDO_LRS.VALID_LRS_PT (point)`
 - Checks if a LRS point is valid (number of dimensions in point is correct). The point passed in must be stored in `SDO_ELEM_INFO` and `SDO_ORDINATES` (not in the `SDO_POINT` field).
- `SDO_LRS.VALID_MEASURE (geom, measure)`
 - Checks if a measure falls within the measure range of a geometric segment.
- `SDO_LRS.CONNECTED_GEOM_SEGMENTS (geom1, geom2)`
 - Checks if two geometric segments are connected.

LRS Application

Transportation Application in the DBMS



ORACLE

12-36

Copyright © Oracle Corporation, 2001. All rights reserved.

Transportation Application in the DBMS

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

Case Study 1

- **Pipeline Asset Management**
 - A pipeline consists of several connected sections
- **System Requirements**
 - Linear reference each section of the pipeline and store associated diameters
 - Linear reference inventory items (that is, valves in the pipeline)

ORACLE

12-37

Copyright © Oracle Corporation, 2001. All rights reserved.

Case Study 1

This is a pipeline asset management case study. The system requirements include the following:

- Store the diameters associated with different sections of the pipe
- Store the location of valves along the pipe

Case Study 1

- Pipeline Sections

Section #	LRS Pipe #	Start M.	End M.	Diameter	...
203	SUS1105	230.56	243.17	0.50	
204	SUS1105	243.17	275.84	0.48	
205	SUS1105	275.84	302.21	0.43	

- Pipeline Valves

LRS Pipe #	Measure	Valve ID	...
SUS1105	230.56	10	
SUS1105	243.17	11	
SUS1105	275.84	12	

Note: No explicit vertices needed to define pipeline sections or valves

ORACLE

12-38

Copyright © Oracle Corporation, 2001. All rights reserved.

Case Study 1 (continued)

This is a simple data model for case study 1. The table that contains the linear-referenced pipe geometries is not shown in the data model on this slide.

The pipeline sections table has a foreign key to the linear-referenced pipe geometry table (that is, the “LRS Pipe #” column). A pipeline section is represented by storing start and end measure values of a linear-referenced pipe geometry (that is, for linear-referenced pipe SUS1105, section 203 starts at measure 230.56, and ends at measure 243.17). The diameter of pipe SUS1105, section 203 is 0.5 meters.

The pipeline valves table has a foreign key to the linear-referenced pipe geometry table (that is, the “LRS Pipe #” column). A valve is represented by a measure value associated with an “LRS Pipe #” (for example, Valve ID 10 is at measure 235.20 of pipe SUS1105).

Note: The pipe geometries are stored only once, and only measure values are used to represent pipe sections and valves.

Case Study 2

- **Traffic Management**
 - Sensors placed on major intersections to monitor traffic congestion
- **System Requirements**
 - Store speed limit, traffic congestion (sensor position), accidents, construction, number of lanes, pavement condition, and so on for a road network

ORACLE

12-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Case Study 2

This is a traffic management case study. The system requirements include the following:

- Store speed limit for a road network
- Store traffic congestion for a road network
- Store accidents for a road network
- Store construction information for a road network

Case Study 2

- **Street Attributes**

Street Name	Speed Limit	LRS_ID	Start M.	End M.	# of Lanes
Main	40	NHL1	0	7.2	2
Main	35	NHL1	7.2	12.4	2
Main	50	NHL1	12.4	16.7	4

- **Traffic Congestion**

Sensor #	Street Name	LRS_ID	Loc. (M)	# of Cars/Hr
3	Main	NHL1	5.5	550
4	Main	NHL1	14.8	883

ORACLE

12-40

Copyright © Oracle Corporation, 2001. All rights reserved.

Case Study 2 (continued)

This is a simple data model for case study 2. The table that contains the linear-referenced street geometries is not shown in the data model on this slide.

The street attribute table has a foreign key to the linear-referenced street geometry table (that is, the “LRS_ID” column). A street name, speed limit and lanes is represented by storing start and end measure values of a linear-referenced street geometry (that is, for linear-referenced street NHL1, Main street has 2 lanes and starts at measure 0, and ends at measure 7.2).

The traffic table has a foreign key to the linear-referenced street geometry table (that is, the “LRS_ID” column). A sensor is represented by a measure value associated with an “LRS_ID” (for example, sensor 3 on Main street averages 550 cars per hour and is located at measure 5.5 of street NHL1).

Note: The street geometries are stored only once, and only measure values are used to represent street attributes and sensors.

LRS Example

- Create a table with a set of linear features

```
-- Create a table with a set of roads (Highways in NH, clipped at the NH
-- border)
create table LRS_HIGHWAYS as
select b.highway,
       sdo_geom.sdo_intersection(a.geom, b.geom, 0.5) geom
from   geod_states a,
       geod_interstates b
where  a.state_abrv = 'NH'
       and sdo_relate(b.geom, a.geom,
                      'querytype=window mask=ANYINTERACT') = 'TRUE';

-- Insert metadata data for a 2D geometry
insert into user_sdo_geom_metadata values (
  'LRS_HIGHWAYS', 'GEOM',
  mdsys.sdo_dim_array (
    mdsys.sdo_dim_element ('Long', -180, 180, 0.5),
    mdsys.sdo_dim_element ('Lat', -90, 90, 0.5)), 8307);
```

ORACLE

12-41

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Example

This set of examples show how to implement create and use LRS data with SQL.

First create a table called LRS_HIGHWAYS that contains all the highways in New Hampshire, clipped at the NH border (because NH only cares about roads that lie inside it's borders). The state of NH geometry is pulled out of the GEOD_STATES table and used as a window against the GEOD_INTERSTATES table, so all highways in the GEOD_INTERSTATES table that have any interaction with the state of New Hampshire are returned. For each highway returned, SDO_GEOM.SDO_INTERSECTION clips the highway at the NH boundary.

At this point, the data in the LRS_HIGHWAYS is not linear referenced. It simply contains all the highways in NH, clipped at the NH state boundary.

Insert metadata in the USER_SDO_GEOM_METADATA view for the LRS_HIGHWAYS table just created.

LRS Example

- Convert non-LRS geometries to LRS geometries

```
-- Converts one geometry at a time; user_sdo_geom_metadata not modified
-- automatically
select SDO_LRS.CONVERT_TO_LRS_GEOM(a.geom)
from lrs_highways a
where a.highway = 'I293';

-- Converts the entire layer and updates user_sdo_geom_metadata for the layer.
-- Make sure there are no spatial indexes when this operation is done
declare
    status varchar2(32);
begin
    status := SDO_LRS.CONVERT_TO_LRS_LAYER('LRS_HIGHWAYS','GEOM',0,1000000,0.5);
    dbms_output.put_line(status);
end;
/
```

ORACLE

12-42

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Example (continued)

Non-LRS geometries can be converted one-at-a-time with the `SDO_LRS.CONVERT_TO_LRS_GEOM` function or an entire layer of non-LRS geometries can be converted with `SDO_LRS.CONVERT_TO_LRS_LAYER`.

All of the non-LRS geometries in the `LRS_HIGHWAYS` table need to be converted to LRS geometries.

The first SQL statement in this slide shows how to linear reference highway 'I293'. The result is not stored anywhere, and no metadata is inserted into `USER_SDO_GEOM_METADATA`.

The second SQL statement in this slide shows how to linear reference the entire `LRS_HIGHWAYS` table previously created. `SDO_LRS.CONVERT_TO_LRS_LAYER` updates the `USER_SDO_GEOM_METADATA` view with the values provided in the procedure for the measure dimension. It also updates each non-LRS geometry in the column specified with LRS geometries.

The lower and upper bounds provided in `SDO_LRS.CONVERT_TO_LRS_LAYER` for the measure dimension are inserted into `USER_SDO_GEOM_METADATA`, but they are not currently used by Oracle Spatial (no integrity checks are made against the measure lower and upper bounds). The tolerance value is important, and used by some LRS functions (that is, `SDO_LRS.CONNECTED_GEOM_SEGMENTS`).

Note: If the layer that is being converted to linear referenced geometries is spatially indexed, drop the spatial index prior to calling `SDO_LRS.CONVERT_TO_LRS_LAYER`.

LRS Example

- Store pavement conditions in an attribute table

```
-- Create a table for storing the pavement condition with a
-- foreign key to the LRS_HIGHWAYS table.
-- A start measure, end measure, and the pavement condition
-- between those measures is stored for each highway
create table pavement_condition (
    highway          varchar2(35),
    from_measure     number,
    to_measure       number,
    condition         varchar2(6));
```

ORACLE

12-43

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Example (continued)

Create an attribute table that contains the pavement condition associated with highway segments.

The table pavement condition stores a foreign key (HIGHWAY), back to the LRS_HIGHWAYS table. Pavement conditions are represented by storing the FROM_MEASURE and TO_MEASURE values associated with a highway, and the associated pavement condition for the highway segment.

Note: The highway geometry points are not repeated in the PAVEMENT_CONDITION table. Only measure values are stored.

LRS Example

- Perform LRS queries

```
-- Return the length of each highway
select highway,
       SDO_LRS.GEOM_SEGMENT_END_MEASURE(geom)
from lrs_highways;

-- Using the pavement condition table, clip and return the road segments
-- where the pavement condition is poor
select SDO_LRS.CLIP_GEOM_SEGMENT(a.geom, b.from_measure, b.to_measure),
       a.highway
from lrs_highways a,
     pavement_condition b
where b.condition = 'poor'
     and a.highway = b.highway;
```

ORACLE

12-44

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Example (continued)

The first query in this slide returns the length of each highway in the LRS_HIGHWAYS table. The SDO_LRS.GEOM_SEGMENT_END_MEASURE function will correspond to the highway length (in meters) for this example because the LRS_HIGHWAYS had a geodetic SRID when it was linear referenced with SDO_LRS.CONVERT_TO_LRS_LAYER.

Dynamic segmentation dynamically locates events along a geometry (possibly for display). The second query is a great example of performing dynamic segmentation with a single SQL statement.

In the WHERE clause all the rows that contain b.condition = 'poor' are selected from the pavement_condition table and joined back to the LRS_HIGHWAYS table to get the corresponding linear-referenced geometry (a.highway = b.highway).

In the SELECT clause, the highway name is selected, and the linear-referenced geometry is clipped with SDO_LRS.CLIP_GEOM_SEGMENT from 'from_measure' to 'to_measure' for each row that contains 'poor' as a condition in the pavement_condition table.

LRS Example

- Perform LRS queries

```
-- Given the measure value of a mile post (for example 3000), return the
-- corresponding long/lat value.
select SDO_LRS.LOCATE_PT(a.geom, 3000, 0)
from lrs_highways a
where highway = 'I293';

-- A car has slid off of a highway. The driver subscribed to rescue service that can
-- track her location through her cell phone by generating a longitude/latitude.
-- Find the highway milepost nearest to the car.
select SDO_LRS.FIND_MEASURE(a.geom,
                             mdsys.sdo_geometry(2001, 8307, NULL,
                             mdsys.sdo_elem_info_array(1,1,1),
                             mdsys.sdo_ordinate_array(-71.48, 43)))
from lrs_highways a
where highway = 'I293';

-- Note: FIND_MEASURE returns the measure associated with the closest point
-- of an LRS geometry to a specified point (in this case, the cell phone location).
```

ORACLE

12-45

Copyright © Oracle Corporation, 2001. All rights reserved.

LRS Example (continued)

The first SQL statement in this slide uses `SDO_LRS.LOCATE_POINT` to return the longitude/latitude values associated with measure 3000 of highway 'I293'. The point returned will be stored in the `SDO_ELEM_INFO` and `SDO_ORDINATE` arrays (not the `SDO_POINT` field).

The second SQL statement demonstrate how real life application can solve problems with a simple query that utilizes Oracle LRS. The scenario is a car slips on some ice and runs off a highway into a ditch. The driver subscribed to a rescue service that can track her location through her cell phone (that is, generate the longitude/latitude for her cell phone position). Either the driver could tell the service provider that she is along highway 'I293', or if the driver has no idea where she is, the provider can find the nearest highway to her cell phone position with a simple spatial query.

Once the provider knows the highway ID, it can use `SDO_LRS.FIND_MEASURE` to locate what measure position the customer is along the highway. `SDO_LRS.FIND_MEASURE` does exactly what `SDO_LRS.PROJECT_POINT` does, but it returns only the measure value instead of the projected point. Now the service provider can rescue the customer by sending a tow truck to the location where the customer is.

Summary

In this lesson, you should have learned:

- **LRS concepts**
- **How to use the functions available in LRS**
- **How to implement linear referencing in Oracle Spatial**

ORACLE

13

Other Features

ORACLE®

Copyright © Oracle Corporation, 2001. All rights reserved.

Objectives

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

- **Describe and use Workspace Manager**
- **Understand basic raster/image operations are under consideration**
- **Understand geocoding**

ORACLE

13-2

Copyright © Oracle Corporation, 2001. All rights reserved.

Overview

This lesson describes additional capabilities associated with Oracle Spatial.

Workspace Manager

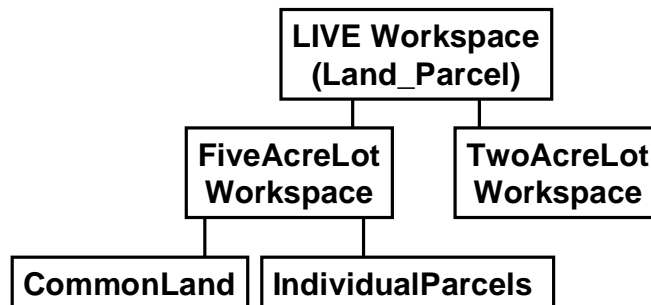
ORACLE

13-3

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is Workspace Manager?

- Enables Web and application based collaboration on database-backed projects
- Provides shareable workspaces within Oracle9i to version data
- Example application: managing parcels in a site development



ORACLE

13-4

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace Manager

Workspace Manager is currently installed with the Oracle9i database, and is also available through OTN with 8.1.7. Database Workspace Manager enables developers to save multiple copies of their transactionally consistent work environment. Workspace Management capabilities were previously found only in Content Management and specialized applications and are not available in IBM DB2 UDB or Microsoft SQL Server.

Example

This example illustrates Workspace Manager collaboration by using shareable workspaces to version land parcel data in a land development scenario.

The example shows a hierarchy of parent and child workspaces. LIVE workspace is the production data. FiveAcreLot and TwoAcreLot workspaces are created as children of the LIVE workspace. Users can work concurrently in both child workspaces and simultaneously version the same data from LIVE to perform tasks like:

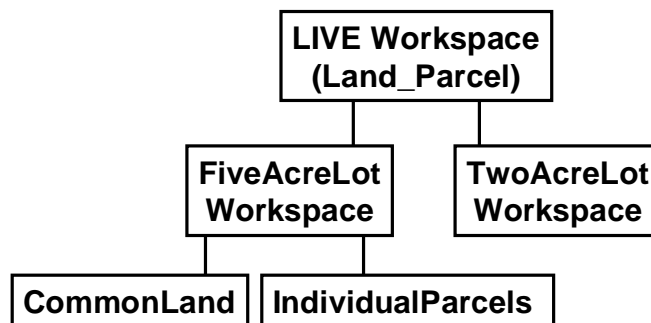
- Customizing site development plans based on lot size
- Determining which scenario will provide the highest profit potential

Workspaces for CommonLand (to model the community pool, playground, and so on) and IndividualParcels (to model lots to build homes on) are children of the FiveAcreLot workspace and could be used to determine the best plot layouts in each of the given scenarios.

What Is a Database Workspace?

A shared, transactionally consistent view of all database tables:

- **Captures changes to version-enabled tables as new row versions within the workspace**
- **Makes versioned rows invisible outside the workspace until explicitly merged with the parent workspace**



ORACLE

13-5

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspaces

A workspace is a virtual environment that provides a transactionally consistent snapshot of the entire database. One or more users share this environment to version data in the database.

The unit of versioning is the table. When a user in a workspace updates a row in a version-enabled table a new version of the row is created. Versions are only accessible within the workspace until explicitly merged with the parent workspace. This also applies to inserts made within a workspace to a version-enabled table.

There can be one or more versions of a row in a workspace from one or more version-enabled tables. The current or active version of a row in a workspace refers to the version of a row to which changes are currently being made.

Workspaces (continued)

Example

The example illustrates how workspaces can be used to manage long duration projects, in this case, land development.

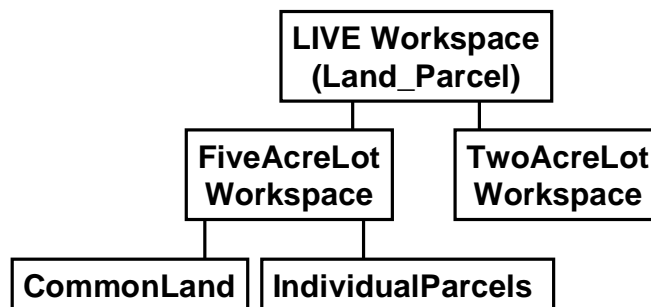
Users in the `FiveAcreLot` workspace can update any content from the `LIVE` workspace. Workspace Manager automatically creates new versions of the rows in `FiveAcreLot`. Meanwhile, data in `LIVE` is still unchanged and accessible to other workspaces and production systems.

Placing the user in a workspace determines their data context. The data context for users in the `CommonLand` workspace includes:

- Latest version of rows that were updated in the `FiveAcreLot` workspace.
- Latest committed data for rows in `LIVE` belonging to tables that are not version-enabled.
- Data for rows in `LIVE` belonging to version-enabled tables, as the data existed when the `FiveAcreLot` workspace was created. When a workspace is refreshed in `LIVE`, it makes the latest committed data for these rows available to users in the `FiveAcreLot` workspace.

How Does Workspace Manager Work?

- Automatically installed with Oracle9i
- Version-enable some or all tables
- Automatically versions only changed rows
- Merges changes with parent to resolve conflicts



ORACLE

13-7

Copyright © Oracle Corporation, 2001. All rights reserved.

How Does Workspace Manager Work?

Versioning

Database Workspace Manager selectively version-enables some or all tables in an existing or new database.

All changes made in a workspace are made by conventional short transactions. Creating an explicit savepoint causes a new version of a row to be created the next time the row is updated. There can be a hierarchy of workspaces in the database. By default, when a workspace is created, it is created from the topmost, or LIVE, database workspace.

Changes made in one or more workspaces to the same production data are captured automatically as new versions of the data. Storage expansion and row proliferation is minimized by versioning only changed rows.

Conflicts are detected automatically before changes are merged into the LIVE workspace and can be resolved by the user with Oracle Enterprise Manager or programmatically through the API.

How Does Workspace Manager Work? (continued)

Example

The example illustrates the use of workspaces in a land development scenario. Because workspaces are transparent to applications, the same SQL can be used against both versioned and nonversioned tables. To implement this workspace environment:

The required tables are first version-enabled.

Then workspaces are created, with different workspaces for each data model

Users

- Login to the database, execute a `GotoWorkspace` procedure, and edit data.
- View the changes in conjunction with data in the parent.
- Merge the edits with the parent. Optionally, other workspaces can be refreshed with the updated information.

Workspace Manager Benefits: Applications

- **Improves concurrency and collaboration for long-duration design projects such as GIS, Web design, and engineering design**
- **Provides auditing to:**
 - **Rollback changes**
 - **View data as it existed at a specific point in time**
- **Manages multiple versions of the data for what-if analysis**

ORACLE

13-9

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace Management Benefits: Applications

- Workspaces improve concurrent access to data in the database
 - They allow any row to be versioned simultaneously in one or more workspaces. Conflicts are detected automatically before changes are merged and can be resolved by the user with Oracle Enterprise Manager or programmatically through the API. This can be particularly useful for collaborative, long-duration projects in areas like web design, engineering design, and Geographic Information (GIS) Systems. In a database without workspaces, users who want to change the same record are serialized by means of locks.
- Workspaces create audit trails that enable users to rollback changes and view data as it existed at a specific time
 - Versions created in a workspace can be time stamped when they are created. Workspace Manager allows users to rollback changes to an earlier version and to compare various versions. Intermediate versions can be retained when the changes in the workspace are merged with the live data.
- Workspaces make it easier to perform multiple what-if analyses
 - Each analysis can create variations on the live data in a different workspace. After the analyses are complete, the results can be stored in the database for quick lookup.

Workspace Manager Benefits: Developer and DBA

- **Rich concurrency and security model**
- **Complete set of workspace semantics**
- **Efficient storage of versioned data**
- **Requires no changes to SQL DML**
- **Easy to manage using Oracle Enterprise Manager**
- **Tightly integrated with Oracle DBMS referential integrity, locking, triggers, import and export**

ORACLE

13-10

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace Management Benefits: Developer and DBA

Workspaces improve concurrent access to data in the database. They allow any row to be versioned simultaneously in one or more workspaces with security and locking while the live data continues to be accessed.

Conflicts are detected automatically before changes are merged and can be resolved by the user with Oracle Enterprise Manager or programmatically through the API.

The Workspace Manager API is implemented as a set of PL/SQL procedures covering all aspects of Workspace use and management for collaboration. A Java API is also available.

Workspace Manager minimizes storage expansion and row proliferation by versioning only changed rows.

Because workspace versions are created implicitly and transparent to the application, it requires no changes to application Data Manipulation Language (DML) SQL. As we'll see, the Workspace API calls are added to the application to navigate and manage the workspace. Workspace Manager is integrated with OEM, providing a powerful graphical interface to manage all aspects of the workspace and resolve version conflicts for rows changed in multiple workspaces.

Database Workspace Manager is tightly integrated with the Oracle9i engine with full support for referential integrity, locking, triggers, import and export.

Workspace Manager Primitives

- **Workspace:** create, refresh, merge, rollback, remove
- **Savepoints:** implicit and explicit savepoints cause a new version to be created when row is updated
- **Privileges:** access, create, delete, rollback, merge
- **Access Modes:** read, write, management, none
- **Locks:** exclusive and shared
- **Differences:** compares savepoints and workspaces
- **Conflict Resolution:** merge child into parent

ORACLE

13-11

Copyright © Oracle Corporation, 2001. All rights reserved.

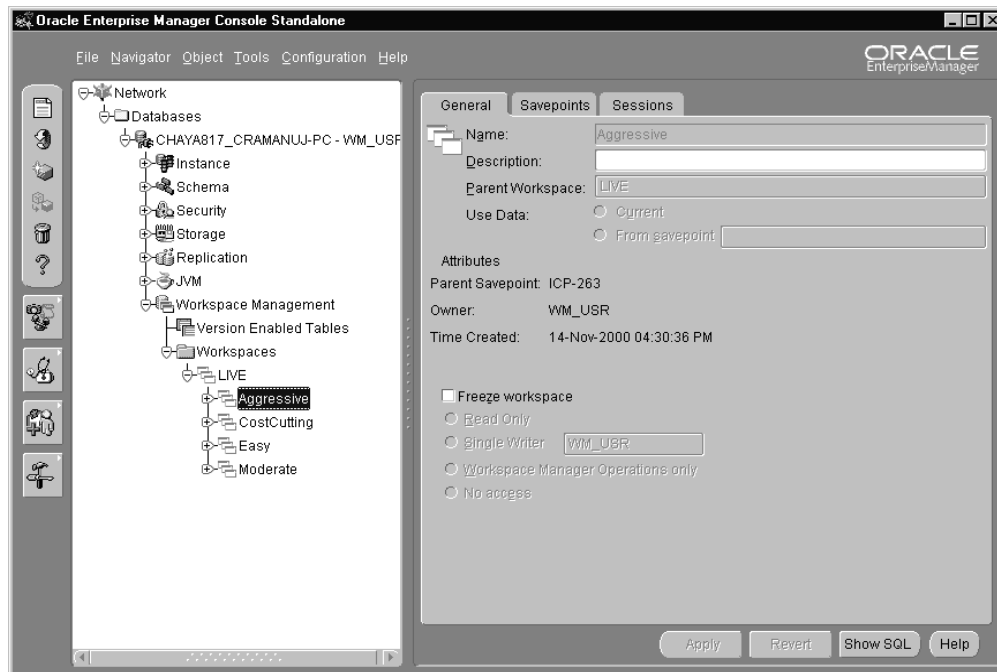
Workspace Manager Primitives

- Workspace data can be refreshed from the parent, merged back to the parent and rolled back to a previous savepoint. In addition they can be created and deleted.
- Savepoints mark a point in time to which operations in the workspace can be rolled back. Implicit savepoints are created by the system when a new child workspace is created. Explicit savepoints are created by the user. A savepoint causes a new version of a row to be created the next time the row is updated. Changes made to the row since an explicit save point can be rolled back.
- Privileges enable users to access, create, delete, rollback and merge workspaces. The WM_ADMIN_ROLE given to the DBA enables all operations. Privileges are either system-wide or workspace-specific. Workspace Manager privileges are integrated with Security Manager.
- Access mode sets a workspace to be one of the following: no access; read only; single writer; management operations only. No access is the default.
- In addition to locks provided by conventional Oracle short transactions, Workspace Manager provides two types of version locks. These locks are primarily intended to eliminate row conflicts between a parent workspace and a child workspace.

Workspace Manager Primitives (continued)

- Differences can exist between any two workspaces, including LIVE or between two savepoints in a given workspace. Row status is either: unchanged, updated, deleted, inserted or nonexistent. Nonexistent means a row with a specific primary key doesn't exist in this workspace or in the base (common ancestor) workspace but was inserted in another workspace. Therefore it is nonexistent in this workspace.
- Conflict exists when two workspaces change the same row. They must be resolved before a child workspace is merged with the parent. Resolution is done either by table (batch mode) or by resolving each conflict individually. The batch mode choices allow you to resolve all differences in the table by choosing the value in the parent, child, or a common ancestor to the parent and child called the base, or none. None indicates that the conflict will not be resolved and the merge will not complete. To choose among these for each conflict specify "individual" and there will be a single prompt for each conflict.

Enterprise Manager Interface



13-13

Copyright © Oracle Corporation, 2001. All rights reserved.

ORACLE

Enterprise Manager Interface

Database Workspace Manager is integrated in with Oracle's Enterprise Manager (OEM) Console. While connected to a database, you will see a folder called Workspace Management that can be expanded to see two subfolders: Version Enabled Tables and Workspaces.

The OEM Version Enabled tables subfolder allows you to view table status and set tables as version-enabled.

The OEM Workspaces folder allows you to:

Create and view workspace hierarchies and attributes. In the slide four workspaces have been created with variations on the live data for what-if cost cutting analysis. The AGGRESSIVE workspace is currently selected.

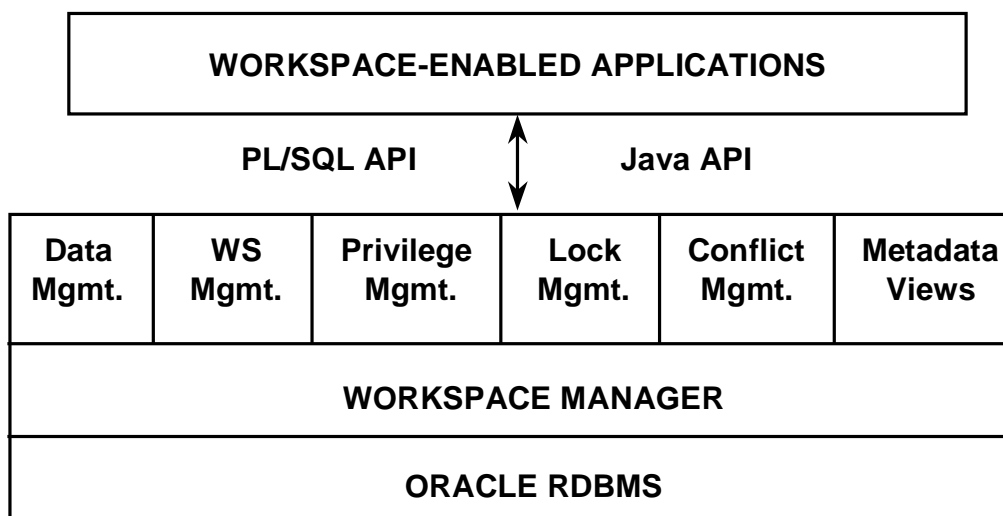
Set and view workspace access modes

- The user access modes for a workspace are:
 - No access, is the default
 - Read only
 - Single writer, allowing all other users to read
 - Workspace operations only, such as merge and rollback

Enterprise Manager Interface (continued)

- Set and view savepoints
 - Implicit savepoint are created by the system when child workspace is created.
 - Explicit savepoint are created by a user.
- Rollback changes since last explicit savepoint
- Resolve differences between any two workspaces or between two savepoints in a workspace.
- Refresh an entire workspace, a table, or rows with data from the parent workspace.
Refreshing a workspace may not succeed if there are conflicts.
- Merge all changes made in the workspace or changes made to a specific table.
- Set privileges to access, create, delete, rollback and merge workspaces.

Architecture



ORACLE

13-15

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace Manager Architecture

Workspace Manager provides PL/SQL procedures that are called by users/applications in order to use the product's features. The procedures are in a single PL/SQL package, but they can be logically grouped into the following categories.

- Table Management
- Workspace Management
- Savepoint Management
- Privilege Management
- Lock Management
- Conflict Management

Database Workspace Manager is tightly integrated with the Oracle9i engine and provides full support for referential integrity, locking, triggers, import and export.

Workspace Manager Administrator Role

- In Oracle9i, Oracle installer automatically installs Workspace Manager and creates the WM_ADMIN_ROLE
- The WM_ADMIN_ROLE role has all Workspace Manager privileges
- By default, the DBA role is granted the WM_ADMIN_ROLE
- The DBA can grant Workspace Manager privileges or can grant the WM_ADMIN_ROLE role to one or more users so they can grant Workspace Manager privileges

ORACLE

13-16

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace Manager Administrator Role

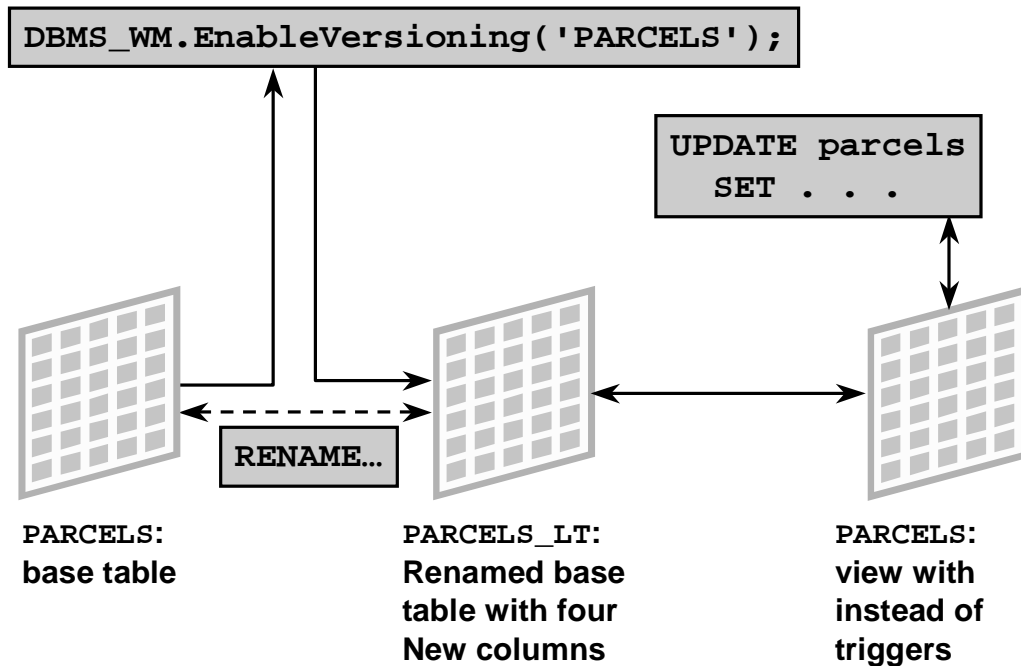
The WM_ADMIN_ROLE role has all Workspace Manager privileges with the grant option.

By default, the DBA role is granted the WM_ADMIN_ROLE.

The DBA or Workspace Manager Admin first needs to determine which users should be granted which privileges. Then, either the DBA can grant the privileges, or the DBA can grant the WM_ADMIN_ROLE role to one or more selected users and these users can grant the appropriate workspace privileges.

Workspace privileges include the ability to ACCESS, CREATE, REMOVE, MERGE, and ROLLBACK a specific workspaces or any workspace.

Workspace-Enable a Table



ORACLE

13-17

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace-Enabling a Table

This example workspace-enables the PARCELS table. EnableVersioning performs the following tasks:

- Augments the table with four workspace metadata columns to store the following data:
 - VERSION_COL is the row version id
 - LTLOCK_COL is the lock status
 - DELSTATUS_COL is the delete status
 - NEXTVER_COL is the next row version id
- Renames the table by adding the suffix _LT to the table name
- Creates a view with the same name as the original table.
- Creates instead of triggers on the view for insert, update, and delete of versioned rows.

Once version-enabled, all rows in the table can support multiple versions of data.

When the view is accessed, it uses the workspace metadata to show only the row versions relevant to the current workspace of the user. The workspace infrastructure is not visible to the end-users.

Workspace-Enabling a Table (continued)

The History Option

This option tracks modifications to table_name using a view named <table_name>_HIST. It must be one of the following values:

NONE: No modifications to the table are tracked. This is the default.

VIEW_W_OVERWRITE: Show only the most recent modification, subsequent changes to a row in the same version overwrite earlier changes.

VIEW_NO_OVERWRITE: Show all modifications, subsequent changes to a row in the same version do not overwrite earlier changes.

Guidelines for Tables Participating in a Workspace

- **Version-enabled table must have a primary key**
- **A table can be version-enabled by the table owner or by a user with WM_ADMIN_ROLE**
- **Tables owned by SYS cannot be version-enabled**
- **Referential integrity constraints are supported on version-enabled tables**
- **Triggers are supported on version-enabled tables with some restrictions**
- **A history option allows the end-user to track all changes made to a version-enabled table**

ORACLE

13-19

Copyright © Oracle Corporation, 2001. All rights reserved.

Guidelines for Tables Participating in a Workspace

Version-enabled tables can have referential integrity constraints, including constraints with the CASCADE and RESTRICT options; however, the following considerations and restrictions apply:

If the parent table in a referential integrity relationship is version-enabled, the child table must be version-enabled also. The child table is the one on which the constraint is defined.

Referential integrity constraints cannot be added when a table is version-enabled; they must be defined before a table is version-enabled.

A child table in a referential integrity relationship is allowed to be version-enabled without the parent table being version-enabled.

A version-enabled table cannot be both a child and a parent in a referential integrity relationship, unless it is a self-referential constraint. That is, the same table can be both the parent and child table in a referential integrity relationship.

Guidelines for Tables Participating in a Workspace (continued)

Version-enabled tables can have triggers defined; however, the following considerations and restrictions apply:

The triggers must be defined before the table is version-enabled.

Only row level triggers are supported. Statement level triggers are not supported.

Only whole row triggers are supported, that is, before and after update triggers for specific columns are not supported.

Triggers on nested table columns are not supported.

The only call-out supported is to PL/SQL procedures. That is, the ACTION_TYPE must be PL/SQL.

Any triggers that are not supported for version-enabled tables are deactivated when versioning is enabled, and are activated when versioning is disabled.

Disabling Workspace Participation for a Table

- **Reverses workspace enabling:**

```
DBMS_WM.DisableVersioning( 'PARCELS' );
```

- **It can be done by the table owner or by a user with the WM_ADMIN_ROLE**
- **Workspace hierarchy and savepoints remain for other version enabled tables**
- **The latest version of each row in LIVE workspace remains**
- **The optional FORCE argument allows the user to disable versioning on a table even if workspaces have modified data in the table**

ORACLE

13-21

Copyright © Oracle Corporation, 2001. All rights reserved.

Disabling Workspace Participation for a Table

The DisableVersioning procedure is used to reverse the effect of the EnableVersioning procedure. It deletes the Workspace Manager infrastructure for versioning of rows, but does not affect any user data in the LIVE workspace. The workspace hierarchy and any savepoints still exist, but all rows are the same as in the LIVE workspace. If there are multiple versions of a row in the table for which versioning is disabled, only the most recent version of the row is kept.

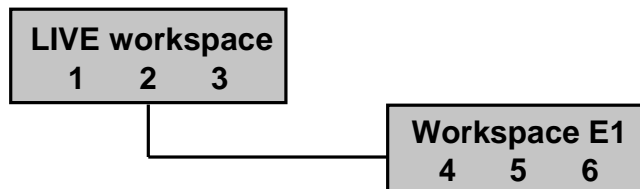
The given example disables versioning on the PARCELS table.

Create a Workspace

- Create a workspace named E1:

```
DBMS_WM.CreateWorkSpace( 'E1' );
```

- New workspace is a child of the current workspace
- Creates an implicit savepoint in the current workspace
- Example: Implicit savepoint 2 is created with workspace E1



ORACLE

13-22

Copyright © Oracle Corporation, 2001. All rights reserved.

Creating a Workspace

If the session has not explicitly entered a workspace, it is in the LIVE database workspace, and the new workspace is a child of the LIVE workspace.

This procedure does not implicitly go to the workspace created. To go to the workspace, use the GotoWorkSpace procedure.

Savepoints are presented in more detail later, but a snapshot known as an implicit savepoint is created in the current version of the current workspace when a new workspace is created. This allows the child workspace to have a transactionally consistent view of the database. The current version for which the savepoint is created does not have to be the latest version in the current workspace.

An exception is raised if a workspace already exists or the user does not have the privilege to create a workspace.

Assign Workspace: Associate a User

- At login, the user is placed in the **LIVE** workspace
- **GOTOWORKSPACE** procedure moves the current user session to the destination workspace
- To include the user in the **E1** workspace:
 - All subsequent modifications to data by the user take place on the latest version in the **E1** workspace

```
DBMS_WM.GotoWorkSpace( 'E1' );
```

ORACLE

13-23

Copyright © Oracle Corporation, 2001. All rights reserved.

Associating a User with a Workspace

When a user logs into Oracle, they are placed in the **LIVE** workspace.

For the user to be placed in the context of a workspace, a **DBMS_WM.GotoWorkSpace** procedure is executed, passing the name of the workspace to go to as a parameter.

Any subsequent changes to data made by the user are made in the context of the latest versions of the workspace. Any changes to tables which do not have versioning enabled are made to the live data.

Because many workspace operations are prohibited when any users are in the workspace, it is often convenient to go to the **LIVE** workspace before performing operations on created workspaces.

To go to the **LIVE** database specify workspace as **LIVE**.

Assign Workspace: Grant Privileges

- **Workspace Privileges:** ACCESS, CREATE, REMOVE, MERGE, and ROLLBACK
- **Privileges in the form of priv_WORKSPACE** allow the user to affect a specified workspace
- **Privileges in the form: priv_ANY_WORKSPACE** allow the user to affect any workspace
- **Example:**

```
DBMS_WM.GrantWorkSpacePriv (
  'ACCESS_WORKSPACE',
  'MERGE_WORKSPACE',
  'E1',
  'SMITH',
  'YES' );
```

ORACLE

13-24

Copyright © Oracle Corporation, 2001. All rights reserved.

Granting Privileges to Users in a Workspace

Workspace Manager implements a set of privileges in addition to standard Oracle database privileges. There are two types of privileges:

- **Workspace-level privileges:** Have names in the form priv_WORKSPACE. They allow the user to affect a specified workspace.
- **System-level privileges:** Have names in the form priv_ANY_WORKSPACE. They allow the user to affect any workspace.

Each privilege can be granted with or without the grant option. The grant option allows the user to which the privilege is granted to grant the privilege to other users.

The example on the slide enables user Smith to access the E1 workspace and merge changes in that workspace, and allows Smith to grant the two specified privileges on E1 to other users.

Assign Workspace: Set Locks

- **Sets session override to exclusive locking**

```
DBMS_WM.SetLockingOn ('E');
```

- **Shared locking on for E1 workspace**

```
DBMS_WM.SetWorkSpaceLockModeOn  
('E1', 'S', TRUE);
```

- **Displays the locking mode in effect for the session**

```
SELECT DBMS_WM.GetLockMode FROM DUAL;
```

- **Locks the row in PARCELS table as shared where
PARCEL_NUMBER = 4 in E1**

```
DBMS_WM.LockRows (  
  'E1', 'PARCELS',  
  'parcel_number = 4', 'S' );
```

ORACLE

13-25

Copyright © Oracle Corporation, 2001. All rights reserved.

Setting Locks in a Workspace

Workspace Manager locks eliminate row conflicts between a parent and child workspace. Locking is enabled at the session level and is a session property independent of the workspace that the session is in. When locking is enabled for a session, it locks rows in all workspaces in which it participates. Locking activities include:

- `SetLockingOn` or `SetLockingOff` sets locking for the session
- `SetWorkspaceLockModeOn` or `SetWorkspaceLockModeOff` sets the default mode for workspace row-level locking. If `OFF`, it enables access to versioned rows in the specified workspace and to corresponding rows in the parent workspace.
- `GetLockMode` returns the locking mode for the parent and child workspace.
- `LockRows` or `UnlockRows` specifies access to versioned rows in a specified table and to corresponding rows in the parent workspace. Either used to proactively lock rows before they are updated or automatically locks row after it is updated by a SQL statement.

In addition to locks provided by conventional Oracle short transactions, Workspace Manager provides two types of version locks:

- Exclusive locks are similar to short transaction locks in that once an exclusive lock has been placed on a record, no other user in the database can change the record except for the session that locked it.
- Shared locks ensure that only users in the workspace in which the row was locked are allowed to modify it.

Create Workspace Savepoint

- **A savepoint is a point in time to which workspace operations can be rolled back.**
- **Examples:**
 - **Create a savepoint named Savepoint1 in the workspace named E1**

```
DBMS_WM.CreateSavePoint ('E1', 'Savepoint1');
```

- **Go to that savepoint**

```
DBMS_WM.GotoSavePoint ('Savepoint1');
```

- **Alter the savepoint description**

```
DBMS_WM.AlterSavePoint ('E1',  
    'Savepoint1', 'Builder 1 parcels split');
```

ORACLE

13-26

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace Savepoints

A savepoint is a point in the workspace to which operations can be rolled back. By creating a savepoint it is possible to prevent any damage to operations performed in the workspace before the savepoint was created. Creating a savepoint causes a new version of a row to be created the next time the row is updated.

There are two types of savepoints:

- Explicit savepoints are created to rollback changes in workspaces.
- Implicit savepoints are created automatically whenever a new workspace is created.

Savepoint Activities

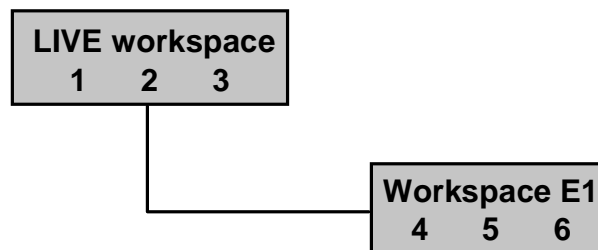
Savepoint activities include:

- Create
- Goto savepoint
- Goto date
- Alter
- Compare difference
- Delete

GotoSavepoint and GotoDate present a read-only view of the workspace at the time of savepoint creation. This is useful for examining the workspace from different savepoints before performing a rollback to a specific savepoint.

Examples of Implicit and Explicit Savepoints

- **Version 2 and an implicit savepoint was created automatically by Workspace Manager when Workspace E1 was created**
- **An explicit savepoint was created on version 5 by a user. This will enable changes in version 6 to be rolled back to version 5**



ORACLE

13-27

Copyright © Oracle Corporation, 2001. All rights reserved.

Examples of Implicit and Explicit Savepoints

Implicit savepoints are created automatically whenever a new workspace is created. In the example, Savepoint 2 was created automatically by Workspace Manager when Workspace E1 was created.

An implicit savepoint is needed so the users in the child workspace get a view of the database that is frozen at the time of workspace creation.

Workspace Manager uses the name `LATEST` to refer to the latest version in the workspace.

There are no explicit privileges associated with savepoints; any user who can access a workspace can create a savepoint in the workspace.

Explicit savepoints can be created and later used to compare differences and effect partial rollbacks in workspaces. In the example, the explicit savepoint created on version 5 allows version 6 changes to be rolled back to version 5.

Compare Savepoints: Find Differences

```
SQL> -- Add rows to difference views
SQL> EXECUTE DBMS_WM.SetDiffVersions
      2 ('CommonLand', 'IndivParcels');
SQL> -- View the rows that were just added
SQL> SELECT ParcelID, ParcelGeom,
      2          wm_diffver, wmcode
      3 FROM Parcel_diff where ParcelID=2;
```

ParcelID	ParcelGeom	WM_DIFFVER	WM_CODE
2	*	FiveAcreLot (base)	NC
2	*	CommonLand, LATEST	U
2	*	IndivParcels, LATEST	U

ORACLE

13-28

Copyright © Oracle Corporation, 2001. All rights reserved.

Compare Savepoints: Find Differences

SetDiffVersions finds differences in values in version-enabled tables for two savepoints in the same workspace or two different workspaces and their common ancestor, or base. It creates row sets in a view that describes the differences between two savepoints and their base. In this example, the differences set consists of:

Row values for the common ancestor to CommonLand and IndivParcels, called the base

Row values for the tables in the first workspace, CommonLand, associated with the latest value or a specified savepoint.

Row values for the tables in the second workspace, IndivParcels, associated with the latest value or a specified savepoint.

You can then select rows from the appropriate view to check comparable table values in the two savepoints and their common base. The view name is xxxx-DIFF, where xxxx is the table name. The status of each row is calculated by analyzing the four metadata columns added to workspace-enable a table. Status values can be either: no change, updated, deleted, inserted, or nonexistent.

Compare Savepoints: Find Differences (continued)

The WMCODE column in the xxxx-DIFF view describes the status of the row and contains one of the following codes: U (updated), D (deleted), I (inserted), NC (no change), NE (nonexistent). NC will appear for rows in workspaces that did not change the value when another workspace did change the value. NE will appear for 'DiffBase' if a row is inserted in one or more of the child workspaces but not in 'DiffBase'.

The example on the slide checks the differences in version-enabled tables for the CommonLand and IndivParcels workspaces. SetDiffVersions writes differences to the difference view, PARCELS_DIFF. The WM_DIFFVER column identifies the row in the differences set. The output has been reformatted for readability.

Note: Changed geometry columns are reported as *

Delete Savepoint

- Delete a savepoint when you no longer need to go to or roll back to it.
- Deleting a savepoint enables you to:
 - Reuse a savepoint name after it is deleted
 - Improve performance of Workspace Manager operations
 - Decrease disk storage used for Workspace Manager structures
- Example: Delete a savepoint named `SAVEPOINT1` in the `E1` workspace

```
DBMS_WM.DeleteSavePoint ('E1', 'SAVEPOINT1');
```

ORACLE

13-30

Copyright © Oracle Corporation, 2001. All rights reserved.

DBMS_WM.DeleteSavePoint

The `DeleteSavePoint` operation removes savepoints explicitly created through workspace manager. This operation can be specified when you no longer need to rollback to a savepoint. Deleting savepoints improves the performance of Workspace Manager as well as decreases the storage requirements. Additionally, it allows the savepoint name to be reused.

Freeze a Workspace

- **Specifies the access allowed to the workspace:**
 - **NO_ACCESS** is the default
 - **READ_ONLY** for all workspace users
 - **1WRITER** enables a single writer, all readers
 - **WM_ONLY** is for Workspace operations
- **Example: Freeze the workspace with 'READ ONLY' access**

```
EXECUTE DBMS_WM.FreezeWorkspace  
( 'E1', 'READ_ONLY' );
```

- **Example: Unfreeze the workspace**

```
EXECUTE DBMS_WM.UnFreezeWorkSpace( 'E1' );
```

ORACLE

13-31

Copyright © Oracle Corporation, 2001. All rights reserved.

Freeze a Workspace

A workspace can be frozen or not frozen. If a workspace is frozen, no changes can be made to data in version-enabled tables, and access to the workspace is restricted.

To make a workspace frozen, use the `FreezeWorkspace` procedure. To make a frozen workspace not frozen, use the `UnfreezeWorkspace` procedure.

In addition, some procedures automatically freeze one or more workspaces.

The `WM_ONLY` option to the `FreezeWorkspace` procedure allows workspace operations such as merge, rollback, refresh, and others while the workspace is frozen.

Rollback a Workspace

- Discards all changes made in the workspace, a table, or after a specified savepoint
- The workspace is not removed
- Rollback all workspace changes:

```
DBMS_WM.RollbackWorkspace( 'TwoAcreLots' );
```

- Rollback changes since a savepoint

```
EXECUTE DBMS_WM.RollbackToSP( 'TwoAcreLots',  
'SVPT1' );
```

- Rollback all changes to a table

```
EXECUTE DBMS_WM.RollbackTable ( 'TwoAcreLots',  
'PARCELS' );
```

ORACLE

13-32

Copyright © Oracle Corporation, 2001. All rights reserved.

Rollback a Workspace

Rolling back a workspace involves deleting either all changes made in the workspace or all changes made since an explicit savepoint. An additionally option is to delete all changes made to a specific versioned table.

Before rolling back to a savepoint:

- Remove any implicit savepoints created since the specified savepoint by merging or removing the descendent workspaces that caused the implicit savepoints to be created.
- Remove all active users.

Rolling back a workspace leaves behind the workspace structure for future use; only the data in the workspace is deleted. To completely remove a workspace, use the RemoveWorkspace procedure.

Refresh a Workspace

- Applies all changes made in the parent to the child since the child was created or last refreshed
- RefreshTable refreshes changes made to a table

```
DBMS_WM.RefreshTable ('E1',  
    'PARCELS', 'zoning = ''Rural''');
```

- RefreshWorkspace refreshes all workspace changes

```
DBMS_WM.RefreshWorkspace('E1');
```

- Before refreshing a table conflicts must be resolved

ORACLE

13-33

Copyright © Oracle Corporation, 2001. All rights reserved.

Refresh a Workspace

The first example refreshes E1 by applying changes made to the PARCELS table where ZONING = 'Rural' in its parent workspace.

The second example refreshes E1 by applying changes made in its parent workspace.

Resolve Workspace Conflicts

- **Conflict exists when the same row is changed in two or more workspaces**
- **Conflicts are detected when a workspace merge or refresh operation is attempted**
- **Conflicts must be resolved before merge or refresh operations succeed**
- **Resolve Conflicts by choosing a row value from:**
 - **BASE**
 - **CHILD**
 - **PARENT**

ORACLE

13-34

Copyright © Oracle Corporation, 2001. All rights reserved.

Resolving Workspace Conflicts

Rows that are changed in the child and parent workspace or in two peer workspaces may lead to data conflicts. Conflicts are discovered during a merge or refresh operation:

- During a merge operation, the changes in a child workspace are incorporated in its parent workspace.
- During a refresh operation, changes made in the parent workspace are incorporated in the child workspace

Conflicts are presented to the user in conflict views, with one conflict view per table. The conflict view lists the primary key of the rows in conflict and also the column values of the rows in the two workspaces that form the conflict.

Conflicts have to be resolved by using the `ResolveConflicts` procedure. During this procedure the user chooses the row value from the base (the common value at the time the child workspace was created), the child, or the parent (where the value changed after the workspace was created) table.

When there are no conflicts between the parent and child workspaces, the data in the two workspaces can be merged.

Conflicts must be resolved before a `MergeWorkspace` or `RefreshWorkspace` operation can be performed.

Conflict Resolution Example: Check for Existence of Conflicts

- Check for conflicts between child and parent

```
DBMS_WM.SetConflictWorkspace  
( 'FIVEACRELOTS' );
```

- View conflicts in PARCELS table

```
Select * from parcels_conf;
```

ORACLE

13-35

Copyright © Oracle Corporation, 2001. All rights reserved.

Conflict Resolution Example: Check for Existence of Conflicts

The general process to check for the existence of conflicts is as follows:

- SetConflictWorkspace: The example checks for any conflicts between child workspace IndividualParcels, and its parent workspace FIVEACRELOTS. The example activates a view for every version-enabled table where the view name is <table_name>_CONF.
- The query on the screen shows the conflicts for that table.

Conflict Resolution Example: Resolve Conflicts

- Start a conflict resolution session

```
DBMS_WM.BeginResolve ('FIVEACRELOTS');
```

- Resolve conflicts by looping through all conflicts

```
EXECUTE DBMS_WM.ResolveConflicts  
  ( 'FIVEACRELOTS', 'PARCELS',  
    'zoning = 'Rural'', 'CHILD');
```

- Keep changes from the preceding step

```
DBMS_WM.CommitResolve ('FIVEACRELOTS');
```

ORACLE

13-36

Copyright © Oracle Corporation, 2001. All rights reserved.

Conflict Resolution Example: Resolve Conflicts

The general process for resolving conflicts is as follows:

BeginResolve: The example starts a conflict resolution session. FiveAcreLots is frozen in 1WRITER mode.

ResolveConflicts: The example resolves conflicts involving rows in the PARCELS table in FiveAcreLots where the value of column ZONING = 'Rural', and uses the values in the child workspace to resolve all such conflicts. It then merges the results of the conflict. The procedure is executed once per affected combination of table and workspace. After each successful execution of ResolveConflicts, a standard database commit operation should be performed. However, any changes are not made permanent in the database until you execute CommitResolve.

CommitResolve: Keeps all the changes from the preceding step. This ends the current conflict resolution session started by BeginResolve and saves all changes in the workspace since the start of the conflict resolution session.

Notw: RollbackResolve discards all the changes from the preceding step. This quits the current conflict resolution session started by BeginResolve, and discards all changes in the workspace since the start of the conflict resolution session.

Merge a Workspace

- Applies changes in a child workspace or table to its parent workspace or table.
- Merge either entire workspace, table or specific rows
- Conflicts must be resolved first.
- MergeTable can optionally rollback to initial workspace state.
- MergeWorkspace can optionally remove the workspace.
- Example: Merge changes in IndividualParcels to its parent workspace (FiveAcreLots) and do not remove IndividualParcels

```
DBMS_WM.MergeWorkspace('IndividualParcels');
```

ORACLE

13-37

Copyright © Oracle Corporation, 2001. All rights reserved.

Merge a Workspace

Merging a workspace applies changes made in a workspace to its parent workspace and optionally removes it. While a merge is in progress the child workspace is frozen in NO_ACCESS mode and the parent workspace is frozen in READ_ONLY mode.

If merging a nonleaf workspace the DROP_WORKSPACE option cannot be used because descendent workspaces still exist.

The example on the slide merges changes in IndividualParcels to its parent workspace and retains workspace IndividualParcels.

Workspace Views

- **Metadata views hold information about:**
 - **Version-enabled tables**
 - **Workspaces**
 - **Savepoints**
 - **Users**
 - **Privileges**
 - **Locks**
 - **Conflicts**
- **Views are read-only to users**
- **Views are used to administer the workspace environment and diagnose problems**

ORACLE

13-38

Copyright © Oracle Corporation, 2001. All rights reserved.

Workspace Manager creates and maintains metadata views to hold information that helps to manage the workspace environment and diagnose problems. These views are read-only to users. Views that span the whole workspace environment are:

- `USER_WM_VERSIONED_TABLES` and `ALL_WM_VERSIONED_TABLES`
- `USER_WM_MODIFIED_TABLES` and `ALL_WM_MODIFIED_TABLES`
- `USER_WORKSPACES` and `ALL_WORKSPACES` contain information about the workspaces a user owns or can access
- `USER_WORKSPACE_SAVEPOINTS` and `ALL_WORKSPACE_SAVEPOINTS`
- `USER_WORKSPACE_PRIVS` and `ALL_WORKSPACE_PRIVS` includes all users' privileges
- `USER_WM_PRIVS` includes privileges the current user has in each workspace
- `ROLE_WM_PRIVS`
- `USER_WM_LOCKED_TABLES` and `ALL_WM_LOCKED_TABLES`
- `DBA_WORKSPACE_USERS` contains user info for workspaces other than `LIVE`
- `USER_WM_RIC_INFO` and `ALL_WM_RIC_INFO` contain referential integrity constraints
- `USER_WM_TAB_TRIGGERS` and `ALL_WM_TAB_TRIGGERS`
- `ALL_VERSION_HVIEW` is a workspace hierarchy

Views created for each workspace enabled table are:

- Conflict view
- Difference view
- Lock view
- History view
- Multiworkspace view

Compress Workspace or Workspace Tree

- **Collapses workspace tree into optimum configuration**
- **Compress when the explicit savepoints in the affected workspaces are no longer needed**
- **Benefits:**
 - **Savepoint names can be reused**
 - **Performance is improved**
 - **Less storage is used**

ORACLE

13-39

Copyright © Oracle Corporation, 2001. All rights reserved.

Compress Workspace/Workspace Tree

Compressing a workspace involves deleting explicit savepoints in the workspace and minimizing the Workspace Manager metadata structures for the workspace. For each range, where a range includes the versions between two consecutive implicit savepoints in a workspace, the latest version of each database row is kept and renumbered to reuse the earliest version in the range.

You can compress a workspace and all its descendent workspaces when the explicit savepoints in the affected workspaces are no longer needed. For example, when you will not need to go to or roll back to any of the savepoints.

A workspace cannot be compressed if there are any sessions with an open regular transaction, or if any user has executed a GotoDate operation specifying a savepoint in the workspace.

Benefits of Compressing a Workspace

The compression operation is useful for the following reasons:

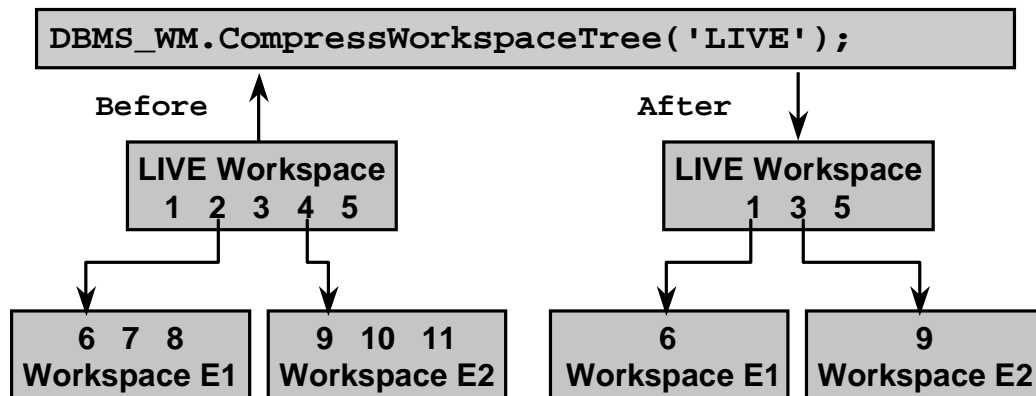
Savepoint names can be reused after they are deleted. A savepoint cannot be created that has the same name as an existing savepoint.

Run-time performance for Workspace Manager operations is improved.

Less disk storage is used for Workspace Manager structures.

Compress Workspace Tree Example

- **Compress LIVE and descendants**
- **Keep versions 8 and 11 renumbered as 6 and 9**



ORACLE

13-40

Copyright © Oracle Corporation, 2001. All rights reserved.

Compress Workspace Tree Example

As stated in the last slide, for each range, where a range includes the versions between two consecutive implicit savepoints in a workspace, the latest version of each database row is kept and renumbered to reuse the earliest version in the range.

In the example, the implicit savepoints in LIVE are 2 and 4. These are the versions upon which workspaces E1 and E2 are based respectively.

The savepoint ranges are 1-2, 3-4 and 5, in LIVE workspace, 6-8 in E1 workspace and 9-11 in E2.

When the versions are compressed the following actions are taken:

Workspace	Range	Action
LIVE	1-2	Savepoint 2 is kept and renamed as 1
LIVE	3-4	Savepoint 4 is kept and renamed as 3
LIVE	5	Savepoint 5 is kept and not renamed
E1	6-8	Savepoint 8 is kept and renamed as 6
E2	9-11	Savepoint 11 is kept and renamed as 9

Other Workspace Tasks

- **Change the workspace description**

```
DBMS_WM.AlterWorkspace  
('E1', 'Test scenario E1');
```

- **Remove a workspace or workspace tree**

```
DBMS_WM.RemoveWorkspaceTree ('E1');
```

- **Check for active sessions**

```
SELECT  
DBMS_WM.IsWorkspaceOccupied('FiveAcreLots')  
FROM DUAL;
```

- **Return the current session workspace**

```
SELECT DBMS_WM.GetWorkspace FROM DUAL;
```

ORACLE

13-41

Copyright © Oracle Corporation, 2001. All rights reserved.

Other Workspace Tasks

`AlterWorkspace` allows the user to change the description of a given workspace. In the slide, the example modifies the description of the `E1` workspace.

`RemoveWorkspace` or `RemoveWorkspaceTree` removes the specified workspace and all its descendant workspaces. The data in the workspaces is rolled back and the workspace structure is removed. Removes all support structures created for the workspace. The workspace ceases to exist. The example in the slide removes the `E1` workspace and all its descendent workspaces.

`IsWorkspaceOccupied` checks whether or not a given workspace has any active sessions. The example in the slide checks if any sessions are active in the `FiveAcreLots` workspace.

`GetWorkspace` returns the current state of session attributes. The example in the slide displays the workspace that the current user is in.

Image/Raster Data Support

ORACLE

13-42

Copyright © Oracle Corporation, 2001. All rights reserved.

Basic Image/Raster Storage and Retrieval

- **In development**
- **Currently accepting requirements**
- **Some planned features include**
 - **Image Storage**
 - Store as tiles**
 - Spatial Indexing capabilities**
 - **Image Retrieval**
 - Based on a query window**
 - Retrieve images or parts of images**

ORACLE

13-43

Copyright © Oracle Corporation, 2001. All rights reserved.

Image/Raster Additions

Adding Image/Raster storage and retrieval in an integrated fashion with Oracle Spatial is currently in development.

This slide should not be construed as a commitment for any product deliverables.

Requirements for raster/imagery functionality are currently being gathered.

Some core database operations with regards to raster/image data is anticipated, including storage, indexing, and retrieval of image/raster data.

Geocoding

ORACLE

13-44

Copyright © Oracle Corporation, 2001. All rights reserved.

What Is Geocoding?

- **The process of associating a spatial location (coordinates) to a street address**
- **Can have varying levels of precision: street level, zip code level, town level, and so on**
- **Oracle does not supply a geocoder, but integrates tightly with any vendor's geocoder**

ORACLE

13-45

Copyright © Oracle Corporation, 2001. All rights reserved.

What is Geocoding?

Geocoding is the process of taking a street address (in text format), and returning a spatial location (the coordinates) associated with that address.

Geocoding can be done at various levels of precision. Some geocoders return the coordinates where an address falls on a street. Others may return the centroid of the zip code boundary, or town boundary the address resides in.

Oracle does not supply a geocoder, but integrates tightly with any vendor's geocoder. Use the vendor's native API to geocode an address. Then take the longitude/latitude values returned from the vendor's API, and place them in an `MDSYS.SDO_GEOMETRY` constructor.

The geometry can then be inserted into a table, or used as a query window.

Summary

In this lesson, you should have learned:

- **How to complete Workspace Manager operations**
- **That basic raster image storage and retrieval is being investigated**
- **Oracle integrates with other vendor's geocoders**

ORACLE

Exercises

Oracle9i Spatial
Release 9.0.1

November 2001, Oracle Spatial Training Guide

Release 9.0.1

Copyright © Oracle Corporation, 1999, 2000, 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, and as set forth in subparagraph (c) (1) (ii) of DFARS 252.227-7013, Rights in Technical Data and Computer Software (October 1988).

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).

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

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to the Spatial Solutions Group, Oracle Corporation, 196 Van Buren Street, Herndon, Virginia 20170. Oracle Corporation does not warrant that this document is error free.

SQL*Loader and SQL*Plus are registered trademarks of Oracle Corporation. Oracle9, and Oracle9i Spatial are trademarks of Oracle Corporation.

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

1. Introduction to Oracle Spatial

No practice.

2. Oracle Spatial Concepts

No practice.

3. Creating Tables with Spatial Layers

- a. Create the following tables:

GEOD_CITIES

Column name	Datatype	Description
LOCATION	SDO_GEOMETRY	City location
CITY	VARCHAR2(42)	City name
STATE_ABRV	VARCHAR2(2)	State code
POP90	NUMBER	Population (1990)
RANK90	NUMBER	Population ranking (1990)

GEOD_INTERSTATES

Column name	Datatype	Description
HIGHWAY	VARCHAR2(6)	Interstate name
GEOM	SDO_GEOMETRY	Interstate geometry

- b. Insert metadata for the following layers into the USER_SDO_GEOM_METADATA view:

- GEOD_CITIES (location)
- GEOD INTERSTATES (geom)

Note about the SQL script provided for exercise 3b:

- The metadata for GEOD_CITIES (location) has a NULL SRID. In an upcoming exercise, you will learn how to associate an SRID with a layer that doesn't have one.
- The tolerance for GEOD_CITIES (location) is set to 0.0000005, which is an example of how tolerances were set prior to Oracle9i. In Oracle9i the tolerance unit for geodetic layers is specified in meters. In a later exercise, the tolerance of GEOD_CITIES (location) will be modified to 0.5 (half a meter), a

more appropriate geodetic tolerance for the GEOD_CITIES (location) layer in Oracle9i.

4. Coordinate Systems Overview

No practice.

5. Loading Spatial Layers

1. The GEOD_CITIES and GEOD_INTERSTATES tables were created in a previous exercise. Now:

- Load the GEOD_CITIES table with the provided control file.
- Load the GEOD_INTERSTATES table with the provided control file.

Note associated with the geod_cities.ctl control file:

In the GEOD_CITIES.ctl control file, the SDO_SRID field is set to NULL for each geometry. As discussed in the note on exercise 3b, in a later exercise how to associate a SRID to a layer that does not have one will be demonstrated.

2. a. Import the GEOD_COUNTIES table from the geod_counties.dmp file.
Use the command: **imp scott/pw4scott file=geod_counties.dmp full=y**

GEOD_COUNTIES

Column name	Datatype	Description
COUNTY	VARCHAR2(31)	City name
FIPSSTCO	VARCHAR2(5)	FIPS county code
STATE	VARCHAR2(30)	State name
STATE_ABRV	VARCHAR2(2)	State code
FIPST	VARCHAR2(2)	FIPS state code
LANDSQMI	NUMBER	Land area (square miles)
TOTPOP	NUMBER	Total population
POPSPQMI	NUMBER	Population density
GEOM	SDO_GEOMETRY	County boundary

- b. Insert metadata for the GEOD_COUNTIES (geom) layer into the USER_SDO_GEOM_METADATA view

c. Import the following tables from the proj_data.dmp file:

- PROJ_CITIES
- PROJ_INTERSTATES
- PROJ_COUNTIES
- PROJ_STATES

Use the command: **imp scott/pw4scott file=proj_data.dmp full=y**

d. Insert metadata for the following layers:

- PROJ_CITIES (location)

- PROJ_INTERSTATES (geom)
 - PROJ_CONTIES (geom)
 - PROJ_STATES (geom)
3. Run the shp2sdo utility on the STATES shapefile provided by executing the convert_states_from_shape.bat file. Alternatively, if you decide to run shp2sdo from the command prompt, don't forget to set the lower and upper bound X and Y values.

Use the (.SQL and .CTL) files generated by shp2sdo to create and load the GEOD_STATES table.

Create the GEOD_STATES table and populate the layer metadata.

SQL> @GEOD_STATES.sql

From the DOS prompt, load the GEOD_STATES table as follows:

- **cd <directory that contains GEOD_STATES.ctl file>**
- **SQLLDR scott/pw4scott GEOD_STATES.ctl**

Note: The shp2sdo utility always loads data in the pre-Oracle Spatial 8.1.6 format (single digit SDO_GTYPE, single digit element type for polygons, and polygon rotation is not enforced). In the next exercise, the data is migrated to the Oracle Spatial 8.1.6+ format.

4. Migrate the following layers to the Oracle Spatial 8.1.6+ format.
- GEOD_CITIES (location)
 - GEOD_COUNTIES (geom)
 - GEOD_INTERSTATES (geom)
 - GEOD_STATES (geom)
5. Earlier, you loaded GEOD_CITIES (location) with no SRID. This exercise associates a SRID with the GEOD_CITIES (location) layer.
- For each row in GEOD_CITIES, set the LOCATION column's SDO_SRID=8307 (WGS84).
 - Add the 8307 SRID to the GEOD_CITIES (location) entry of the USER_SDO_GEOM_METADATA view.
 - Modify the tolerance to 0.5 meters for GEOD_CITIES (location)

6. Indexing Spatial Data

Exercises are in the next section.

7. Indexing Spatial Data and Tuning and Administration

1. Create non-spatial indexes on the following columns:

- GEOD_CITIES(CITY)
- GEOD_STATES (STATE)
- GEOD_INTERSTATES (HIGHWAY)
- GEOD_COUNTIES (STATE_ABRV, COUNTY)

Note: The projected layers you imported created their non-spatial indexes during the import:

- PROJ_CITIES(CITY)
- PROJ_STATES (STATE)
- PROJ_INTERSTATES (HIGHWAY)
- PROJ_COUNTIES (STATE_ABRV, COUNTY)

2. Without using the Spatial Index Advisor, estimate the quadtree fixed tiling level for the PROJ_INTERSTATES(GEOM) column using the following criteria:

10000 tiles for extent surrounding all geometries in the layer
(ALL_GID_EXTENT)

Note: In an upcoming exercise, you will use the Spatial Index Advisor to estimate the quadtree fixed tiling level. The Spatial Index Advisor calls ESTIMATE_TILING_LEVEL, and does a lot more.

3. Create a fixed level quadtree spatial index on the PROJ_INTERSTATES (GEOM) column. Use the fixed tiling level determined in question 2, as the SDO_LEVEL.

4. For optimal query performance, statistics (compute or estimate) must be generated on quadtree spatial index tables. Run the SQL command that generates SQL to compute statistics on the spatial index table of the PROJ_INTERSTATES layer.

```
SQL> SELECT 'ANALYZE TABLE ' || sdo_index_table ||  
        ' COMPUTE STATISTICS; '  
FROM user_sdo_index_info  
WHERE sdo_index_type = 'QTREE';
```

Note: Don't forget to run the SQL generated by the SQL statement above, for example,

```
SQL> ANALYZE TABLE <index table name> compute statistics;
```

Note: The following SQL statements show you which spatial index tables have statistics. The last_analyzed column will contain a date if the table is analyzed.

```
SQL> SELECT a.last_analyzed,  
           a.table_name,  
           b.sdo_index_table,  
           b.sdo_index_type  
FROM user_tables a,  
     user_sdo_index_info b  
WHERE a.table_name = b.table_name;
```

5. The Oracle Spatial Index Advisor is a very good tool to help you tune quadtree spatial indexes. But before you use the Spatial Index Advisor, you must grant the “select any dictionary” privilege to the Oracle user you loaded spatial data into.

Log in as the Oracle SYSTEM user (i.e. SQLPLUS system/pw4system) and grant the “select any dictionary” privilege to the Oracle user you loaded your spatial data into. (for example, GRANT select any dictionary TO scott;).

6. To start the Spatial Index Advisor, at the DOS prompt type: oemapp sdoadvisor. Use the Spatial Index Advisor to create quadtree fixed spatial indexes for the following layers:

- PROJ_STATES (geom)
- PROJ_COUNTIES (geom)
- PROJ_CITIES (location) - This is a point only layer.

7. For optimal query performance, statistics (compute or estimate) must be generated on quadtree spatial index tables.

- From a SQL*Plus session, make sure you are connected as the user that owns the data (i.e. scott/pw4scott). You can verify this by typing SHOW USER at the SQL*Plus prompt.
- Compute statistics on the spatial index tables for the following layers that were indexed with quadtree spatial indexes:
 - PROJ_STATES (geom)
 - PROJ_COUNTIES (geom)
 - PROJ_CITIES (location)

8. Go back to the Spatial Index Advisor.

- View the PROJ_STATES(geom) layer as a reference layer. A reference layer will help you locate dense areas in other layers. Select “Zoom to Reasonable Size” to view the PROJ_STATES(geom) layer in an area where the PROJ_COUNTIES layer will be dense (i.e. the center of the US).
- View data and the tiles associated with the PROJ_STATES(geom) layer.
- Make the PROJ_COUNTIES (geom) layer your current layer and Select “Zoom to Reasonable Size”. Your viewport is now zoomed to an area covered by approximately 200 tiles (whose size are the same as the PROJ_COUNTIES(geom) quadtree index tiles).
- The goal for quadtree spatial indexes is to tune them for your dense areas. If you tune well for dense areas, queries against sparse areas will be tuned too. When you “Zoom to Reasonable Size” in a dense area, determine if too much data is displayed. If so, you may want to use a smaller tile size (larger SDO_LEVEL). If not enough data displays when you “Zoom to Reasonable Size” in a dense area, you may want to reindex your data with a larger tile size (smaller SDO_LEVEL).
- Use the Layer->Query menu item to look at the results of both SDO_FILTER and SDO_RELATE operations.

9. Create R-tree indexes on the geodetic layers previously loaded.

8. Spatial Queries

Perform the following queries:

1. Perform a primary filter query (approximation using the index only) to find all the interstate highways that interact with state of New Hampshire.
2. Perform a primary filter query to find an approximation of all the counties that interact with the state of New York.
3. Find the cities in the rectangle which has a lower left coordinate of (-109, 37) and an upper right coordinate of (-102, 40) together with their population.
4. Find the counties that interact with the rectangle (-109, 37) (-102, 40) together with their population.
5. Find the cities within 100 miles of highway 'I10/I5'.
6. Find the 5 cities nearest to highway 'I170' whose population is > 300000, and return their distance to highway 'I170'.
7. Determine the relationship between the state of New Hampshire and its counties (that is, show which counties are border counties and which are not).

9. Spatial Analysis

Perform the following queries:

1. Find the sum of the area in square miles of all of the counties around Passaic county in New Jersey.
2. Same as above, but categorized by state.
3. Find all states with a population density over 500 per square mile, and display the area of each state in square kilometers.
4. Find all the states with borders longer than 1500 miles, and return the length of each border.
5. From the GEOD_STATES(GEOM) layer, generate a geometry represents the union of following states: NH, VT and NY.

6. From the GEOD_STATES(geom) layer, find the centroid of the following New England states: MA, NH, VT, ME, RI, and CT.
7. From the GEOD_STATES(geom) layer, find the geometry that forms a convex hull around the following New England states: MA, NH, VT, ME, RI, and CT.
8. From the PROJ_COUNTIES (geom) layer, calculate the minimum bounding rectangle (MBR) for each of the following states: MA, NH, VT, RI, ME, and CT.

Notes:

- This exercise asks you to build the minimum bounding rectangle for the states mentioned, but you can not use the GEOD_COUNTIES layer. Don't forget to group the result by state (that is, to generate one MBR per state).
- SDO_AGGR_MBR cannot be run against the GEOD_COUNTIES layer because optimized rectangles are not valid for geodetic data.

10. Advanced Coordinate Systems Concepts

No practice.

11. Advanced Spatial Indexing Concepts

Partitioning Exercises

1. A non-partitioned table of fabricated yellow pages data called YELLOW_PAGES has been loaded under the Oracle user Scott.

Create a partitioned table called YELLOW_PAGES_PART by selecting all the data from the YELLOW_PAGES table, and use the CATEGORY column as the partition key for YELLOW_PAGES_PART.

The CATEGORY column can contain the following values. Place each category in its own partition.

- Category 1 - restaurants
- Category 2 - banks
- Category 3 - gas stations
- Category 4 - grocery stores
- Category 5 - hotels
- Category 6 - auto dealers

2. Insert metadata for the partitioned PROJ_PART_COUNTIES(location) column into the USER_SDO_GEOM_METADATA view, and create a partitioned spatial index. This table contains 360600 rows. The spatial index may take about 8 minutes to create.
3. In the current SQL*Plus session, type: **set timing on**. Determine the number of restaurants (CATEGORY=1) that are within 8 miles of the location (-73.8, 40.7) using the YELLOW_PAGES table, and from the YELLOW_PAGES_PART table. Compare the query times on the non-partitioned and partitioned tables.

Note: Try running this test a second time. The first time you ran the test includes the time to cache disk blocks into memory (query returns 9244 rows). The second time you run the test will perform the query from cached disk blocks.

4. Query the YELLOW_PAGES_PART table to the three closest businesses to the location (-73.8042, 40.7613). Display the business name and distance in miles.
5. Query the YELLOW_PAGES_PART table to find the 5 closest THAI restaurants to the location (-73.8042, 40.7613). Display the restaurant name and distance in miles

Function-Based Index Exercises

6. Log in as the Oracle SYSTEM user (i.e. SQLPLUS system/pw4system) and grant the “query rewrite” privilege to the Oracle user you loaded your spatial data into (for example, grant query rewrite to scott).
7. Log back into Oracle as SCOTT (scott/pw4scott). Add two new columns to the GEOD_CITIES table, CITY_LONGITUDE and CITY_LATITUDE, both of type NUMBER.

Populate the newly added columns by pulling the longitude and latitude values out of the LOCATION column of the GEOD_CITIES table.

8. Create a function GET_GEOM, that takes as input arguments a longitude and a latitude, and returns an MDSYS.SDO_GEOMETRY constructor with the SDO_POINT field populated.
9. Insert a row into USER_SDO_GEOM_METADATA for the GET_GEOM function based index.
10. Create a function-based spatial index for the CITY_LONGITUDE and CITY_LATITUDE columns in the GEOD_CITIES table.

11. Alter the current SQL*Plus session to set “query_rewrite_enabled=true” and “query_rewrite_integrity=trusted”.
12. Use the function-based index on the GEOD_CITIES longitude, latitude columns to find all the cities within 100 miles of (-73.94, 40.66).

12. Linear Referencing System (LRS)

1. Using data from the GEOD_INTERSTATES table, create a table LRS_HIGHWAYS that contains highways that interact with the state of Delaware (DE), but clipped at the DE border.

Insert metadata for the new layer you created, LRS_HIGHWAYS(geom).

Create an index on the LRS_HIGHWAYS (highway) column.

2. Linear reference all the geometries in LRS_HIGHWAYS(geom) layer.
3. Find the length of each highway in DE. Specify length in meters.
4. Create a table called PAVEMENT_CONDITION for storing the pavement conditions associated with highway segments in DE.

The PAVEMENT_CONDITION should not have any geometries, just measure values associated with the linear referenced geometries in LRS_HIGHWAYS.

Insert records with highway segment pavement conditions into the PAVEMENT_CONDITION table.

5. Dynamically generate the clipped sections of highways in DE that have a poor pavement condition. This operation is also known as dynamic segmentation.
6. Find the position along the “I95” highway where the measure value is 3000 meters.
7. Find the distance along highway “I295” to the following location (-75.59, 39.70).