Chapitre VIII

Extension images d’ORACLE

http://www.csis.gvsu.edu/GeneralInfo/Oracle/appdev.920/a96630/toc.htm
http://www.infres.enst.fr/~dombd/Doc8i/inter.815/a67295/toc.htm

Extensions d’ORACLE

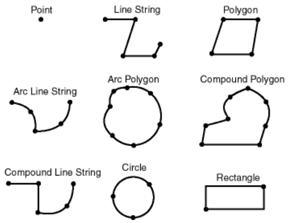
- 8.1 – Concepts spatiaux
- 8.2 – Indexation et interrogation
- 8.3 – Fonctions particulières
- 8.4 – Oracle Spatial 10g
- 8.5 – Conclusions

8.1 – Concepts spatiaux

- Relationel-objet
- Deux dimensions (*x*, *y*)
- Tolérance
- Couches
- Indexation

Types géométriques

- Points et ensembles de points
- Polyligne
- Polygone
- Chaîne d’arcs
- Polygones composés
- Cercles
- Rectangles optimisés



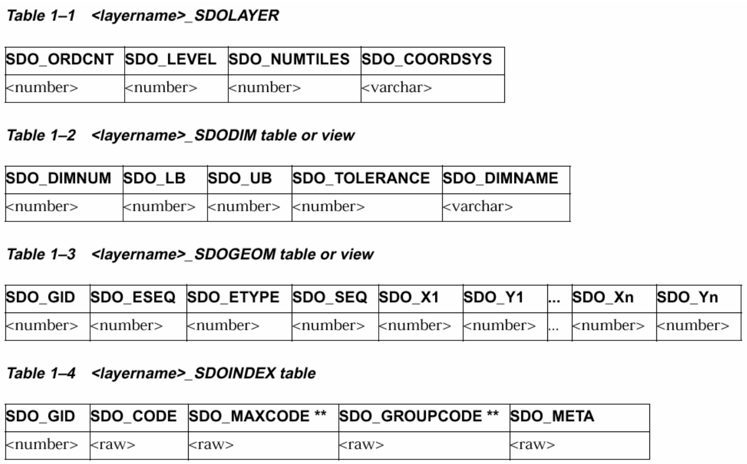


Table 1–1 <layername>_SDOLAYER

SDO_ORDCNT	SDO_LEVEL	SDO_NUMTILES	SDO_COORDSYS
<number>	<number>	<number>	<varchar>

Table 1–2 <layername>_SDODIM table or view

SDO_DIMNUM	SDO_LB	SDO_UB	SDO_TOLERANCE	SDO_DIMNAME
<number>	<number>	<number>	<number>	<varchar>

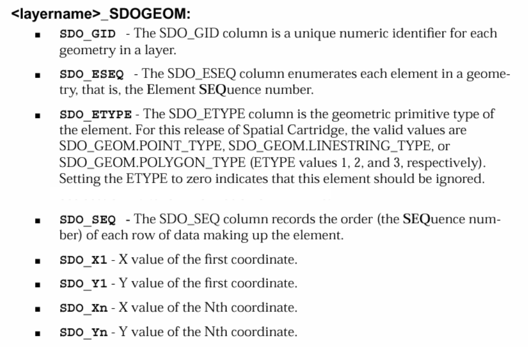
Table 1–3 <layername>_SDOGEOM table or view

SDO_GID	SDO_ESEQ	SDO_ETYPE	SDO_SEQ	SDO_X1	SDO_Y1	...	SDO_Xn	SDO_Yn
<number>	<number>	<number>	<number>	<number>	<number>	...	<number>	<number>

Table 1–4 <layername>_SDOINDEX table

SDO_GID	SDO_CODE	SDO_MAXCODE **	SDO_GROUPCODE **	SDO_META
<number>	<raw>	<raw>	<raw>	<raw>

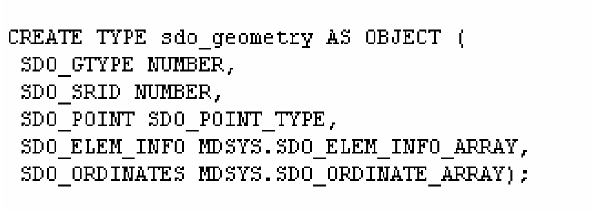
SDO Geom



<layername>_SDOGEOM:

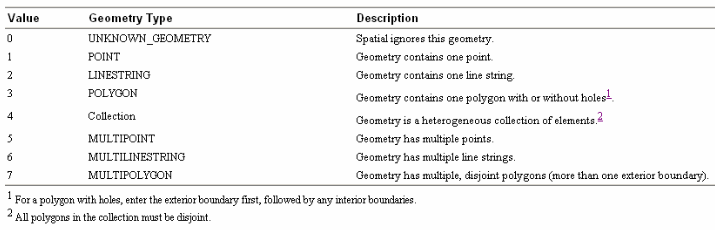
- **SDO_GID** - The SDO_GID column is a unique numeric identifier for each geometry in a layer.
- **SDO_ESEQ** - The SDO_ESEQ column enumerates each element in a geometry, that is, the Element **SEQ**uence number.
- **SDO_ETYPE** - The SDO_ETYPE column is the geometric primitive type of the element. For this release of Spatial Cartridge, the valid values are SDO_GEOM.POINT_TYPE, SDO_GEOM.LINESTRING_TYPE, or SDO_GEOM.POLYGON_TYPE (ETYPE values 1, 2, and 3, respectively). Setting the ETYPE to zero indicates that this element should be ignored.
- **SDO_SEQ** - The SDO_SEQ column records the order (the **SEQ**uence number) of each row of data making up the element.
- **SDO_X1** - X value of the first coordinate.
- **SDO_Y1** - Y value of the first coordinate.
- **SDO_Xn** - X value of the Nth coordinate.
- **SDO_Yn** - Y value of the Nth coordinate.

Création d'objets géométriques



```
CREATE TYPE sdo_geometry AS OBJECT (
 SDO_GTYPE NUMBER,
 SDO_SRID NUMBER,
 SDO_POINT SDO_POINT_TYPE,
 SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,
 SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```

SDO_GTYPE

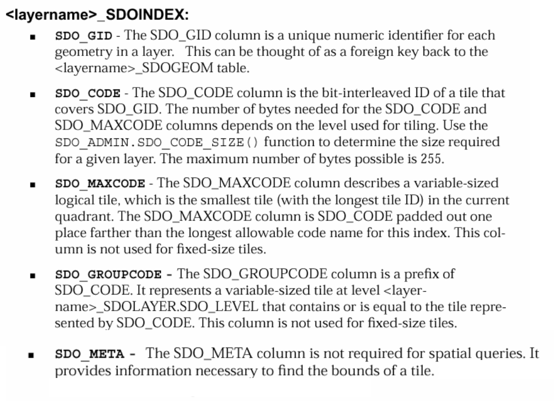


Value	Geometry Type	Description
0	UNKNOWN_GEOMETRY	Spatial ignores this geometry.
1	POINT	Geometry contains one point.
2	LINESTRING	Geometry contains one line string.
3	POLYGON	Geometry contains one polygon with or without holes[1].
4	Collection	Geometry is a heterogeneous collection of elements.[2]
5	MULTIPOINT	Geometry has multiple points.
6	MULTILINESTRING	Geometry has multiple line strings.
7	MULTIPOLYGON	Geometry has multiple, disjoint polygons (more than one exterior boundary).

[1] For a polygon with holes, enter the exterior boundary first, followed by any interior boundaries.

[2] All polygons in the collection must be disjoint.

SDO Index

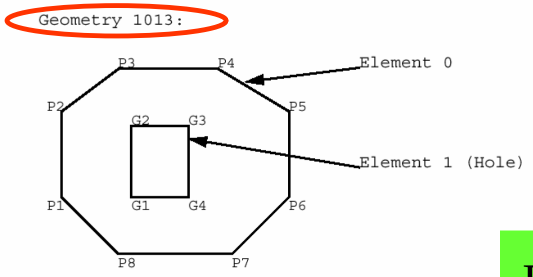


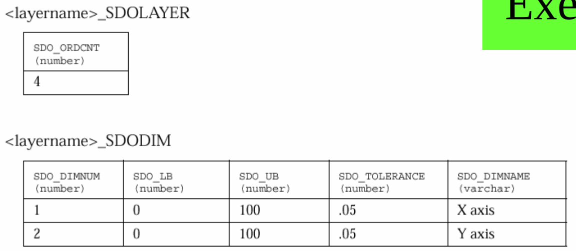
<layername>_SDOINDEX:

- `SDO_GID` - The SDO_GID column is a unique numeric identifier for each geometry in a layer. This can be thought of as a foreign key back to the <layername>_SDOGEOM table.
- `SDO_CODE` - The SDO_CODE column is the bit-interleaved ID of a tile that covers SDO_GID. The number of bytes needed for the SDO_CODE and SDO_MAXCODE columns depends on the level used for tiling. Use the `SDO_ADMIN.SDO_CODE_SIZE()` function to determine the size required for a given layer. The maximum number of bytes possible is 255.
- `SDO_MAXCODE` - The SDO_MAXCODE column describes a variable-sized logical tile, which is the smallest tile (with the longest tile ID) in the current quadrant. The SDO_MAXCODE column is SDO_CODE padded out one place farther than the longest allowable code name for this index. This column is not used for fixed-size tiles.
- `SDO_GROUPCODE` - The SDO_GROUPCODE column is a prefix of SDO_CODE. It represents a variable-sized tile at level <layername>_SDOLAYER.SDO_LEVEL that contains or is equal to the tile represented by SDO_CODE. This column is not used for fixed-size tiles.
- `SDO_META` - The SDO_META column is not required for spatial queries. It provides information necessary to find the bounds of a tile.

Exemple

Figure 1–2 Complex Polygon





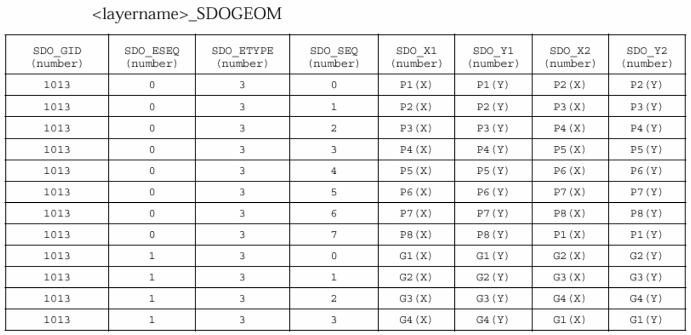
<layername>_SDOLAYER

SDO_ORDCNT (number)
4

<layername>_SDODIM

SDO_DIMNUM (number)	SDO_LB (number)	SDO_UB (number)	SDO_TOLERANCE (number)	SDO_DIMNAME (varchar)
1	0	100	.05	X axis
2	0	100	.05	Y axis

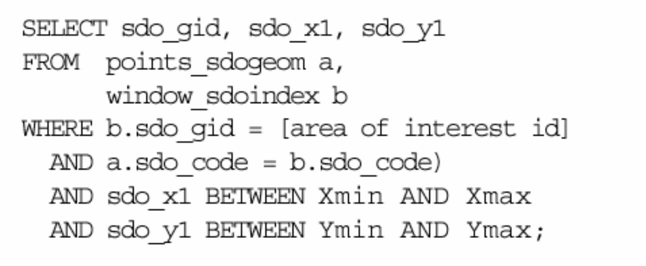
Suite



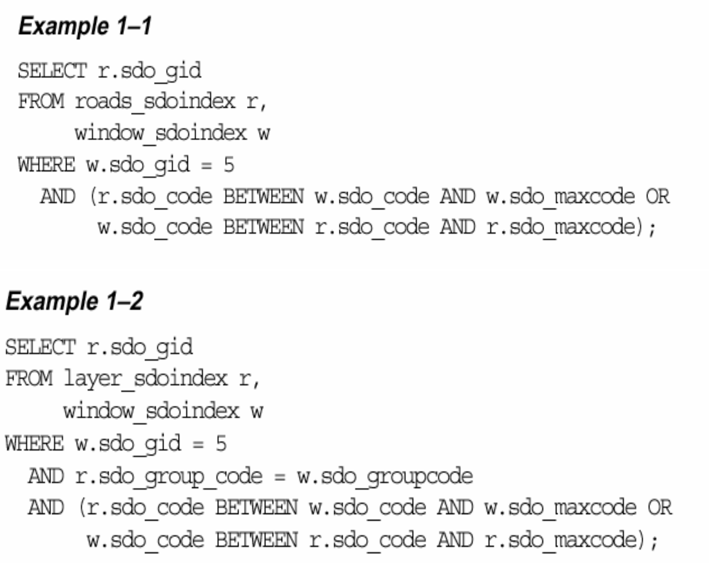
<layername>_SDOGEOM

SDO_GID (number)	SDO_ESEQ (number)	SDO_ETYPE (number)	SDO_SEQ (number)	SDO_X1 (number)	SDO_Y1 (number)	SDO_X2 (number)	SDO_Y2 (number)
1013	0	3	0	P1 (X)	P1 (Y)	P2 (X)	P2 (Y)
1013	0	3	1	P2 (X)	P2 (Y)	P3 (X)	P3 (Y)
1013	0	3	2	P3 (X)	P3 (Y)	P4 (X)	P4 (Y)
1013	0	3	3	P4 (X)	P4 (Y)	P5 (X)	P5 (Y)
1013	0	3	4	P5 (X)	P5 (Y)	P6 (X)	P6 (Y)
1013	0	3	5	P6 (X)	P6 (Y)	P7 (X)	P7 (Y)
1013	0	3	6	P7 (X)	P7 (Y)	P8 (X)	P8 (Y)
1013	0	3	7	P8 (X)	P8 (Y)	P1 (X)	P1 (Y)
1013	1	3	0	G1 (X)	G1 (Y)	G2 (X)	G2 (Y)
1013	1	3	1	G2 (X)	G2 (Y)	G3 (X)	G3 (Y)
1013	1	3	2	G3 (X)	G3 (Y)	G4 (X)	G4 (Y)
1013	1	3	3	G4 (X)	G4 (Y)	G1 (X)	G1 (Y)

Requête



```
SELECT sdo_gid, sdo_x1, sdo_y1
FROM   points_sdogeom a,
       window_sdoindex b
WHERE b.sdo_gid = [area of interest id]
  AND a.sdo_code = b.sdo_code)
  AND sdo_x1 BETWEEN Xmin AND Xmax
  AND sdo_y1 BETWEEN Ymin AND Ymax;
```



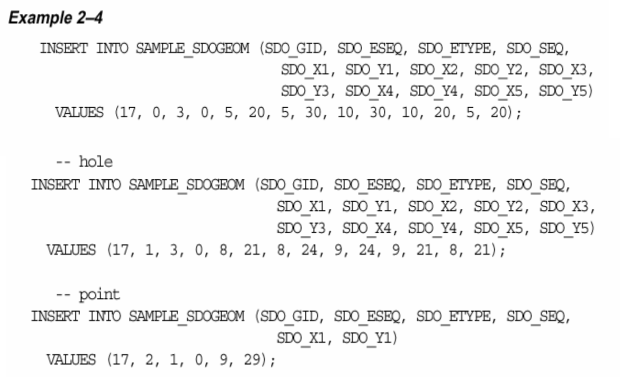
Example 1–1

```
SELECT r.sdo_gid
FROM roads_sdoindex r,
     window_sdoindex w
WHERE w.sdo_gid = 5
  AND (r.sdo_code BETWEEN w.sdo_code AND w.sdo_maxcode OR
       w.sdo_code BETWEEN r.sdo_code AND r.sdo_maxcode);
```

Example 1–2

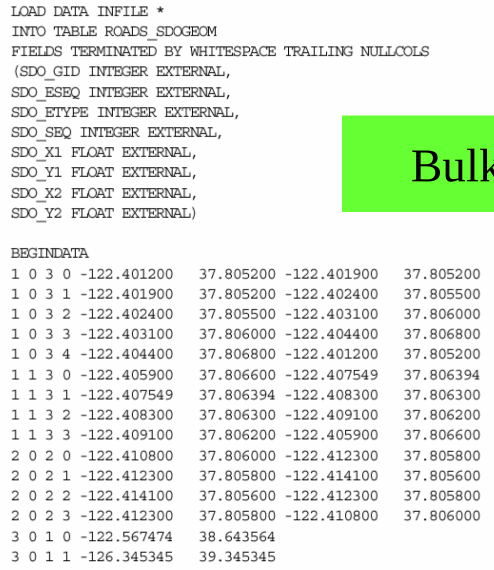
```
SELECT r.sdo_gid
FROM layer_sdoindex r,
     window_sdoindex w
WHERE w.sdo_gid = 5
  AND r.sdo_group_code = w.sdo_groupcode
  AND (r.sdo_code BETWEEN w.sdo_code AND w.sdo_maxcode OR
       w.sdo_code BETWEEN r.sdo_code AND r.sdo_maxcode);
```

INSERT



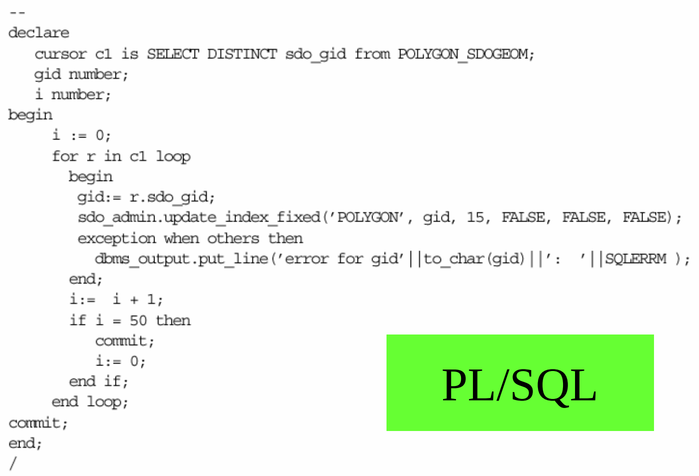
Example 2–4

```
   INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                                 SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                                 SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
    VALUES (17, 0, 3, 0, 5, 20, 5, 30, 10, 30, 10, 20, 5, 20);

    -- hole
  INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                                SDO_X1, SDO_Y1, SDO_X2, SDO_Y2, SDO_X3,
                                SDO_Y3, SDO_X4, SDO_Y4, SDO_X5, SDO_Y5)
   VALUES (17, 1, 3, 0, 8, 21, 8, 24, 9, 24, 9, 21, 8, 21);

    -- point
  INSERT INTO SAMPLE_SDOGEOM (SDO_GID, SDO_ESEQ, SDO_ETYPE, SDO_SEQ,
                                SDO_X1, SDO_Y1)
   VALUES (17, 2, 1, 0, 9, 29);
```



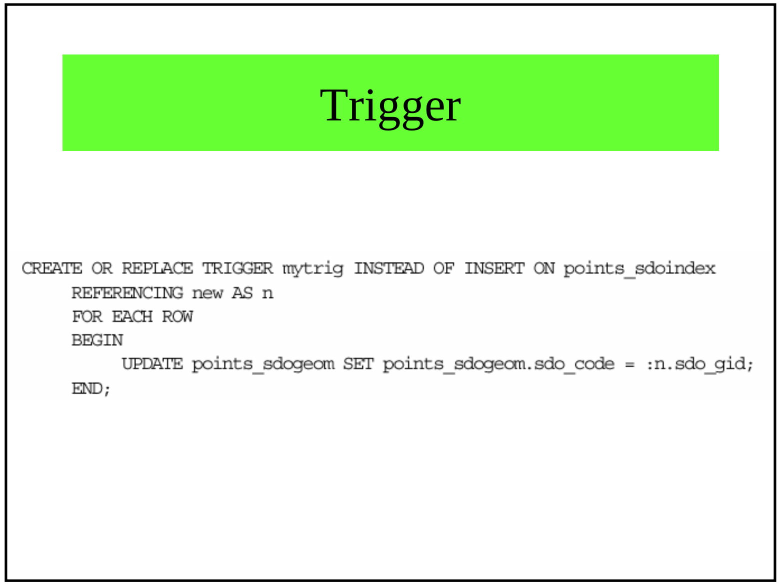
```
LOAD DATA INFILE *
INTO TABLE ROADS_SDOGEOM
FIELDS TERMINATED BY WHITESPACE TRAILING NULLCOLS
(SDO_GID INTEGER EXTERNAL,
SDO_ESEQ INTEGER EXTERNAL,
SDO_ETYPE INTEGER EXTERNAL,
SDO_SEQ INTEGER EXTERNAL,
SDO_X1 FLOAT EXTERNAL,
SDO_Y1 FLOAT EXTERNAL,
SDO_X2 FLOAT EXTERNAL,
SDO_Y2 FLOAT EXTERNAL)

BEGINDATA
1 0 3 0 -122.401200   37.805200 -122.401900   37.805200
1 0 3 1 -122.401900   37.805200 -122.402400   37.805500
1 0 3 2 -122.402400   37.805500 -122.403100   37.806000
1 0 3 3 -122.403100   37.806000 -122.404400   37.806800
1 0 3 4 -122.404400   37.806800 -122.401200   37.805200
1 1 3 0 -122.405900   37.806600 -122.407549   37.806394
1 1 3 1 -122.407549   37.806394 -122.408300   37.806300
1 1 3 2 -122.408300   37.806300 -122.409100   37.806200
1 1 3 3 -122.409100   37.806200 -122.405900   37.806600
2 0 2 0 -122.410800   37.806000 -122.412300   37.805800
2 0 2 1 -122.412300   37.805800 -122.414100   37.805600
2 0 2 2 -122.414100   37.805600 -122.412300   37.805800
2 0 2 3 -122.412300   37.805800 -122.410800   37.806000
3 0 1 0 -122.567474   38.643564
3 0 1 1 -126.345345   39.345345
```

Bulk loading



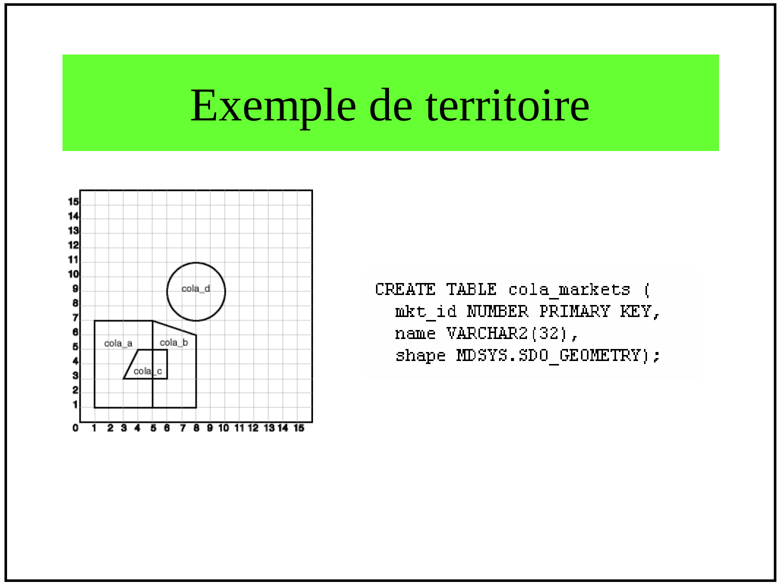
```
--
declare
   cursor c1 is SELECT DISTINCT sdo_gid from POLYGON_SDOGEOM;
   gid number;
   i number;
begin
     i := 0;
     for r in c1 loop
       begin
        gid:= r.sdo_gid;
        sdo_admin.update_index_fixed('POLYGON', gid, 15, FALSE, FALSE, FALSE);
        exception when others then
          dbms_output.put_line('error for gid'||to_char(gid)||':  '||SQLERRM );
       end;
       i:=  i + 1;
       if i = 50 then
          commit;
          i:= 0;
       end if;
     end loop;
commit;
end;
/
```

PL/SQL



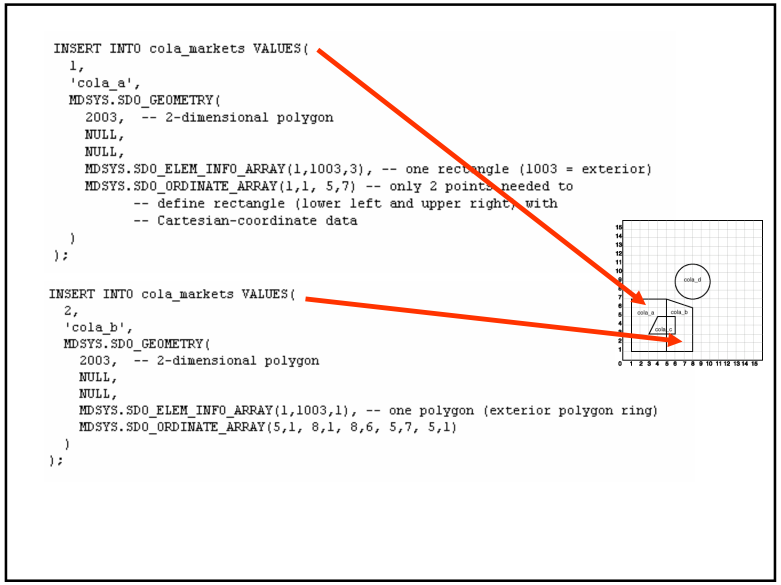
Trigger

```
CREATE OR REPLACE TRIGGER mytrig INSTEAD OF INSERT ON points_sdoindex
      REFERENCING new AS n
      FOR EACH ROW
      BEGIN
            UPDATE points_sdogeom SET points_sdogeom.sdo_code = :n.sdo_gid;
      END;
```

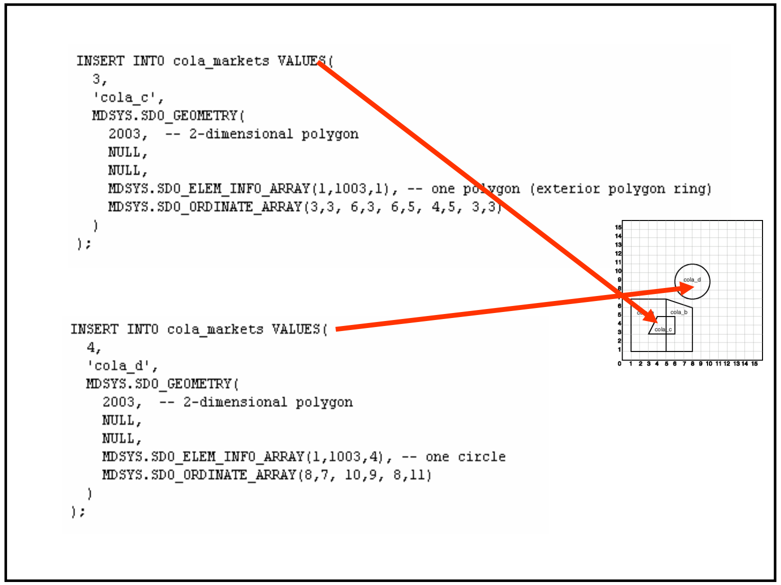


Exemple de territoire

```
CREATE TABLE cola_markets (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape MDSYS.SDO_GEOMETRY);
```



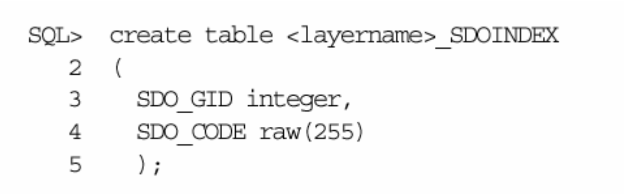
```
INSERT INTO cola_markets VALUES(
  1,
  'cola_a',
  MDSYS.SDO_GEOMETRY(
    2003,  -- 2-dimensional polygon
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
    MDSYS.SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
          -- define rectangle (lower left and upper right) with
          -- Cartesian-coordinate data
  )
);

INSERT INTO cola_markets VALUES(
  2,
  'cola_b',
  MDSYS.SDO_GEOMETRY(
    2003,  -- 2-dimensional polygon
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1), -- one polygon (exterior polygon ring)
    MDSYS.SDO_ORDINATE_ARRAY(5,1, 8,1, 8,6, 5,7, 5,1)
  )
);
```



```
INSERT INTO cola_markets VALUES(
  3,
  'cola_c',
  MDSYS.SDO_GEOMETRY(
    2003,  -- 2-dimensional polygon
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1), -- one polygon (exterior polygon ring)
    MDSYS.SDO_ORDINATE_ARRAY(3,3, 6,3, 6,5, 4,5, 3,3)
  )
);

INSERT INTO cola_markets VALUES(
  4,
  'cola_d',
  MDSYS.SDO_GEOMETRY(
    2003,  -- 2-dimensional polygon
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,4), -- one circle
    MDSYS.SDO_ORDINATE_ARRAY(8,7, 10,9, 8,11)
  )
);
```

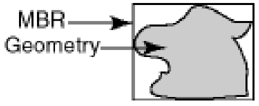
8.2 – Indexation et interrogation

- Quadtree / R-tree

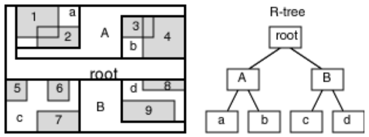


```
SQL>  create table <layername>_SDOINDEX
   2  (
   3    SDO_GID integer,
   4    SDO_CODE raw(255)
   5    );
```

R-tree

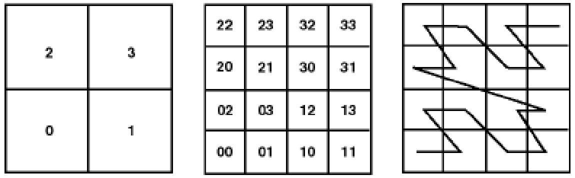


Rectangle englobant



Principe de l'indexation

Quadtree



2	3
0	1

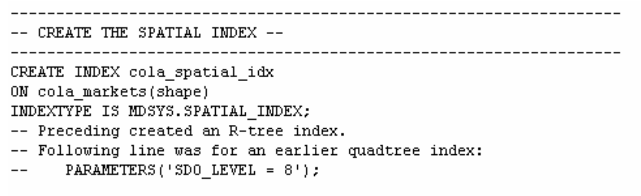
22	23	32	33
20	21	30	31
02	03	12	13
00	01	10	11

Quadtree avec clés de Peano (codage de Morton)

HH codes

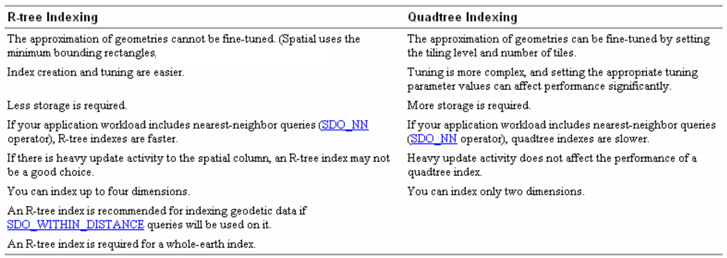
- HHCODEs (Helical Hyperspatial Codes)
- Peano space-filling curves
- Longitude/latitude/altitude/temps

Création d'index



```
--------------------------------------------------------------------
-- CREATE THE SPATIAL INDEX --
--------------------------------------------------------------------
CREATE INDEX cola_spatial_idx
ON cola_markets(shape)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
-- Preceding created an R-tree index.
-- Following line was for an earlier quadtree index:
--    PARAMETERS('SDO_LEVEL = 8');
```

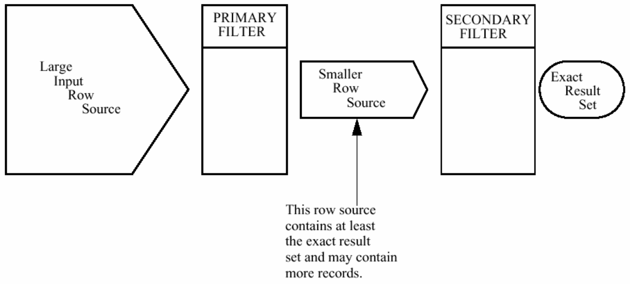
Choisir un type d'index



R-tree Indexing	Quadtree Indexing
The approximation of geometries cannot be fine-tuned. (Spatial uses the minimum bounding rectangles.	The approximation of geometries can be fine-tuned by setting the tiling level and number of tiles.
Index creation and tuning are easier.	Tuning is more complex, and setting the appropriate tuning parameter values can affect performance significantly.
Less storage is required.	More storage is required.
If your application workload includes nearest-neighbor queries (SDO_NN operator), R-tree indexes are faster.	If your application workload includes nearest-neighbor queries (SDO_NN operator), quadtree indexes are slower.
If there is heavy update activity to the spatial column, an R-tree index may not be a good choice.	Heavy update activity does not affect the performance of a quadtree index.
You can index up to four dimensions.	You can index only two dimensions.
An R-tree index is recommended for indexing geodetic data if SDO_WITHIN_DISTANCE queries will be used on it.	
An R-tree index is required for a whole-earth index.	

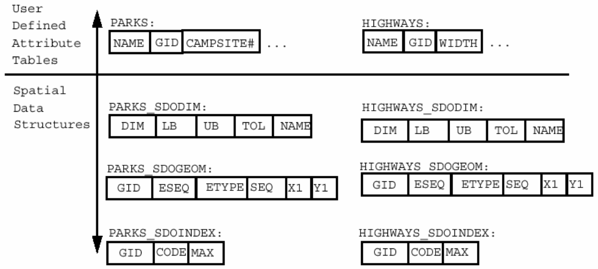
Traitement des requêtes

Figure 3–1 Query Model

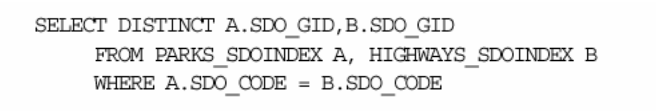


Jointure spatiale

Figure 3–4 Spatial Join of Two Layers

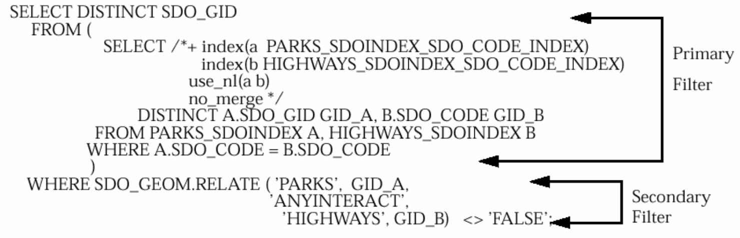


Premier filtrage



```
SELECT DISTINCT A.SDO_GID,B.SDO_GID
      FROM PARKS_SDOINDEX A, HIGHWAYS_SDOINDEX B
      WHERE A.SDO_CODE = B.SDO_CODE
```

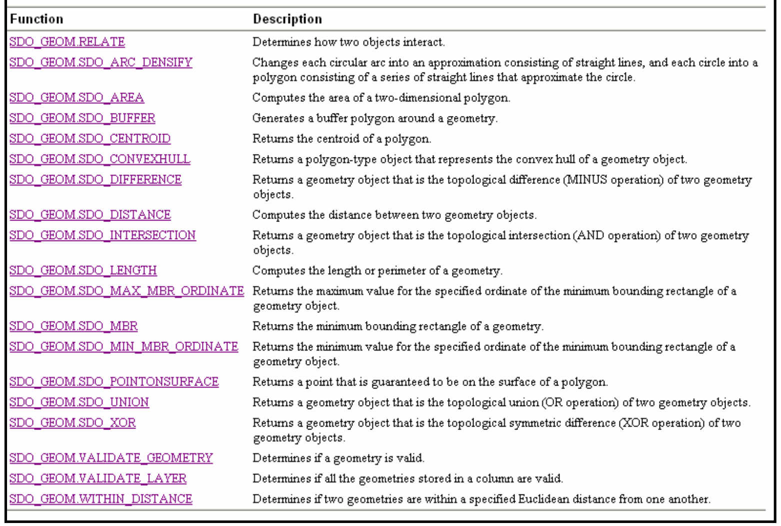
Second filtrage



```
SELECT DISTINCT SDO_GID
   FROM (
            SELECT /*+ index(a  PARKS_SDOINDEX_SDO_CODE_INDEX)
                       index(b HIGHWAYS_SDOINDEX_SDO_CODE_INDEX)
                      use_nl(a b)
                      no_merge */
               DISTINCT A.SDO_GID GID_A, B.SDO_CODE GID_B
           FROM PARKS_SDOINDEX A, HIGHWAYS_SDOINDEX B
          WHERE A.SDO_CODE = B.SDO_CODE
          )
  WHERE SDO_GEOM.RELATE ( 'PARKS',  GID_A,
                          'ANYINTERACT',
                           'HIGHWAYS', GID_B)   <> 'FALSE';
```

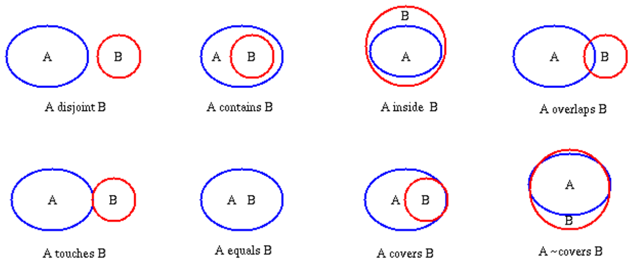
8.3 – Fonctions particulières

- Opérations spatiales classiques
- Relations topologiques

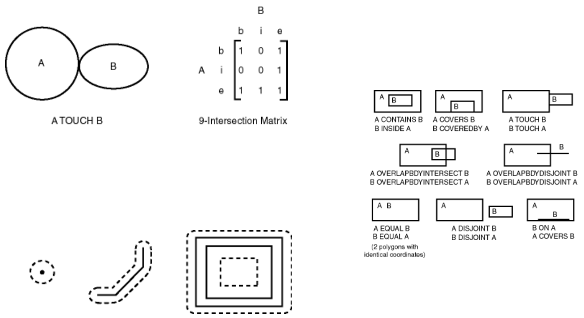


Function	Description
SDO_GEOM.RELATE	Determines how two objects interact.
SDO_GEOM.SDO_ARC_DENSIFY	Changes each circular arc into an approximation consisting of straight lines, and each circle into a polygon consisting of a series of straight lines that approximate the circle.
SDO_GEOM.SDO_AREA	Computes the area of a two-dimensional polygon.
SDO_GEOM.SDO_BUFFER	Generates a buffer polygon around a geometry.
SDO_GEOM.SDO_CENTROID	Returns the centroid of a polygon.
SDO_GEOM.SDO_CONVEXHULL	Returns a polygon-type object that represents the convex hull of a geometry object.
SDO_GEOM.SDO_DIFFERENCE	Returns a geometry object that is the topological difference (MINUS operation) of two geometry objects.
SDO_GEOM.SDO_DISTANCE	Computes the distance between two geometry objects.
SDO_GEOM.SDO_INTERSECTION	Returns a geometry object that is the topological intersection (AND operation) of two geometry objects.
SDO_GEOM.SDO_LENGTH	Computes the length or perimeter of a geometry.
SDO_GEOM.SDO_MAX_MBR_ORDINATE	Returns the maximum value for the specified ordinate of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_MBR	Returns the minimum bounding rectangle of a geometry.
SDO_GEOM.SDO_MIN_MBR_ORDINATE	Returns the minimum value for the specified ordinate of the minimum bounding rectangle of a geometry object.
SDO_GEOM.SDO_POINTONSURFACE	Returns a point that is guaranteed to be on the surface of a polygon.
SDO_GEOM.SDO_UNION	Returns a geometry object that is the topological union (OR operation) of two geometry objects.
SDO_GEOM.SDO_XOR	Returns a geometry object that is the topological symmetric difference (XOR operation) of two geometry objects.
SDO_GEOM.VALIDATE_GEOMETRY	Determines if a geometry is valid.
SDO_GEOM.VALIDATE_LAYER	Determines if all the geometries stored in a column are valid.
SDO_GEOM.WITHIN_DISTANCE	Determines if two geometries are within a specified Euclidean distance from one another.

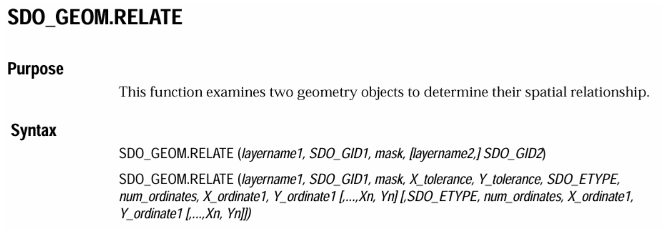
Relations topologiques d’Egenhofer



Exemple de relation topologique



SDO_RELATE



SDO_GEOM.RELATE

Purpose

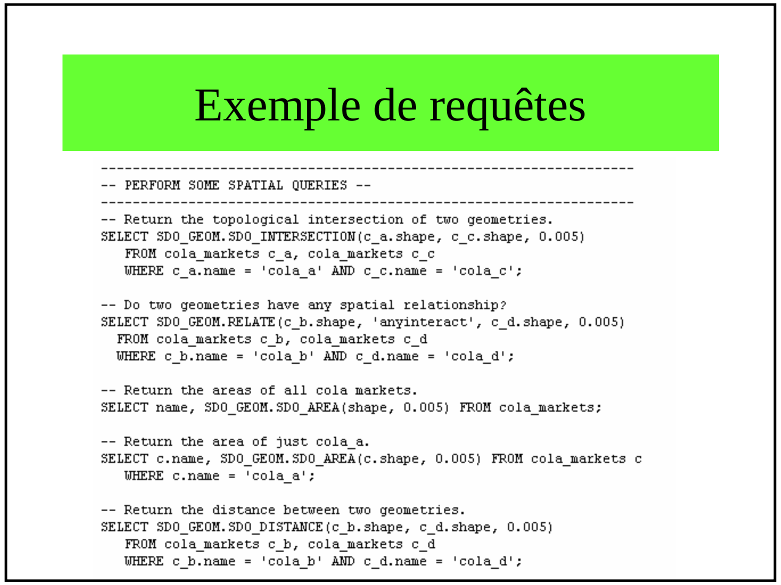
This function examines two geometry objects to determine their spatial relationship.

Syntax

SDO_GEOM.RELATE (*layername1, SDO_GID1, mask, [layername2,] SDO_GID2*)

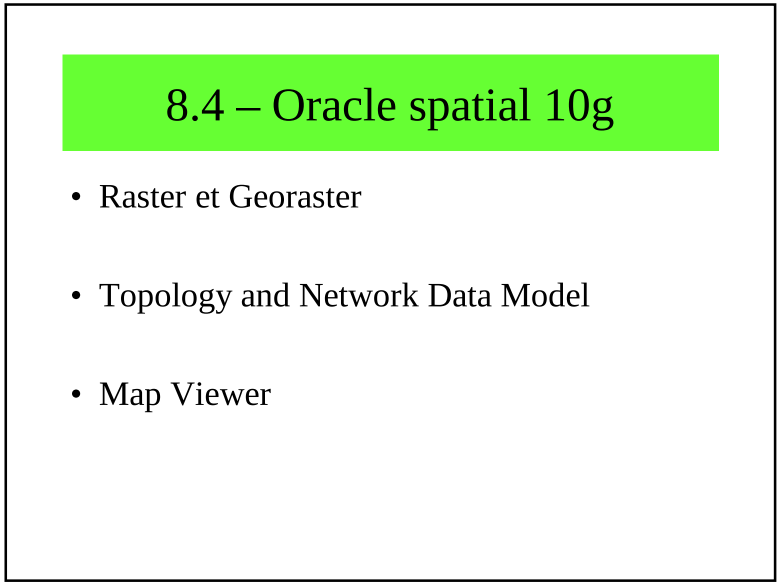
SDO_GEOM.RELATE (*layername1, SDO_GID1, mask, X_tolerance, Y_tolerance, SDO_ETYPE, num_ordinates, X_ordinate1, Y_ordinate1 [,...,Xn, Yn] [,SDO_ETYPE, num_ordinates, X_ordinate1, Y_ordinate1 [,...,Xn, Yn]]*)

- ANYINTERACT - Returns TRUE if the objects are not disjoint.
- CONTAINS - Returns TRUE if the second object is entirely within the first object and the object boundaries do not touch.
- COVEREDBY - Returns TRUE if the first object is entirely within the second object and the object boundaries touch at one or more points.
- COVERS - Returns TRUE if the second object is entirely within the first object and the boundaries touch in one or more places.
- DISJOINT - Returns TRUE if the objects have no common boundary or interior points.
- EQUAL - Returns TRUE if the objects share every point of their boundaries and interior, including any holes in the objects.
- INSIDE - Returns TRUE if the first object is entirely within the second object and the object boundaries do not touch.
- OVERLAPBDYDISJOINT - Returns TRUE if the objects overlap, but their boundaries do not interact.
- OVERLAPBDYINTERSECT - Returns TRUE if the object overlap, and their boundaries intersect in one or more places.
- TOUCH - Returns TRUE if the two objects share a common boundary point, but no interior points.



Exemple de requêtes

```
--------------------------------------------------------------------
-- PERFORM SOME SPATIAL QUERIES --
--------------------------------------------------------------------
-- Return the topological intersection of two geometries.
SELECT SDO_GEOM.SDO_INTERSECTION(c_a.shape, c_c.shape, 0.005)
   FROM cola_markets c_a, cola_markets c_c
   WHERE c_a.name = 'cola_a' AND c_c.name = 'cola_c';

-- Do two geometries have any spatial relationship?
SELECT SDO_GEOM.RELATE(c_b.shape, 'anyinteract', c_d.shape, 0.005)
  FROM cola_markets c_b, cola_markets c_d
  WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';

-- Return the areas of all cola markets.
SELECT name, SDO_GEOM.SDO_AREA(shape, 0.005) FROM cola_markets;

-- Return the area of just cola_a.
SELECT c.name, SDO_GEOM.SDO_AREA(c.shape, 0.005) FROM cola_markets c
   WHERE c.name = 'cola_a';

-- Return the distance between two geometries.
SELECT SDO_GEOM.SDO_DISTANCE(c_b.shape, c_d.shape, 0.005)
   FROM cola_markets c_b, cola_markets c_d
   WHERE c_b.name = 'cola_b' AND c_d.name = 'cola_d';
```



8.4 – Oracle spatial 10g

- Raster et Georaster
- Topology and Network Data Model
- Map Viewer

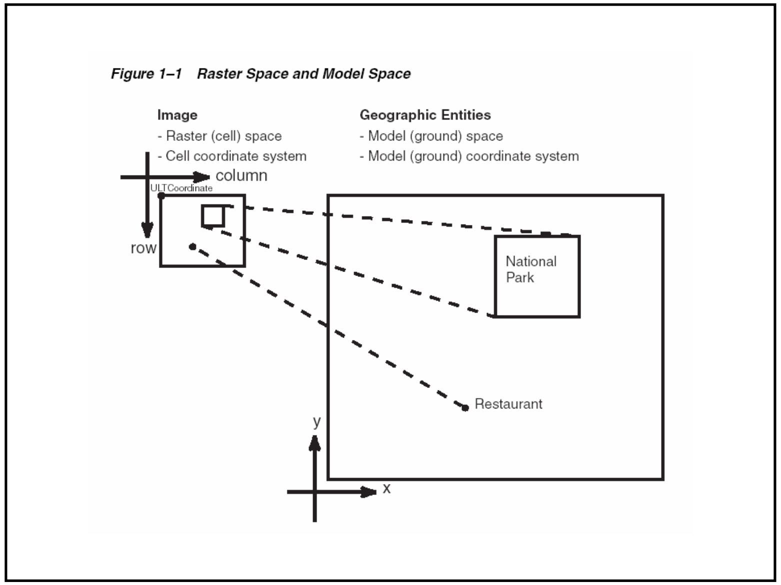


Figure 1–1 Raster Space and Model Space

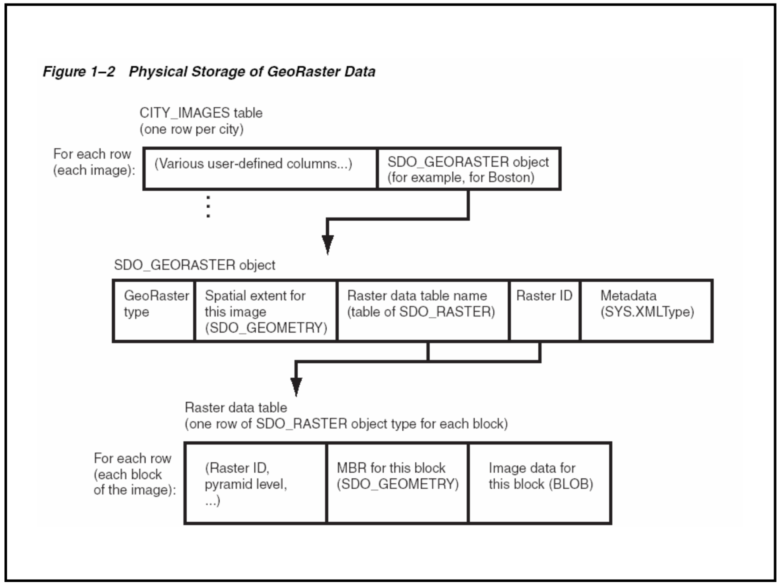


Figure 1–2 Physical Storage of GeoRaster Data

CITY_IMAGES table
(one row per city)

For each row (each image):

(Various user-defined columns...)	SDO_GEORASTER object (for example, for Boston)

SDO_GEORASTER object

GeoRaster type	Spatial extent for this image (SDO_GEOMETRY)	Raster data table name (table of SDO_RASTER)	Raster ID	Metadata (SYS.XMLType)

Raster data table
(one row of SDO_RASTER object type for each block)

For each row (each block of the image):

(Raster ID, pyramid level, ...)	MBR for this block (SDO_GEOMETRY)	Image data for this block (BLOB)

Figure 1–3 GeoRaster Data in an Oracle Database

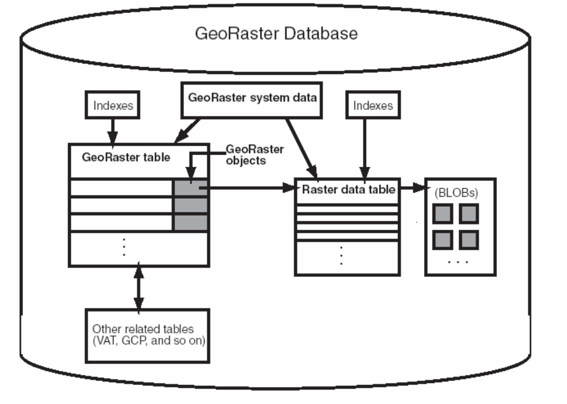
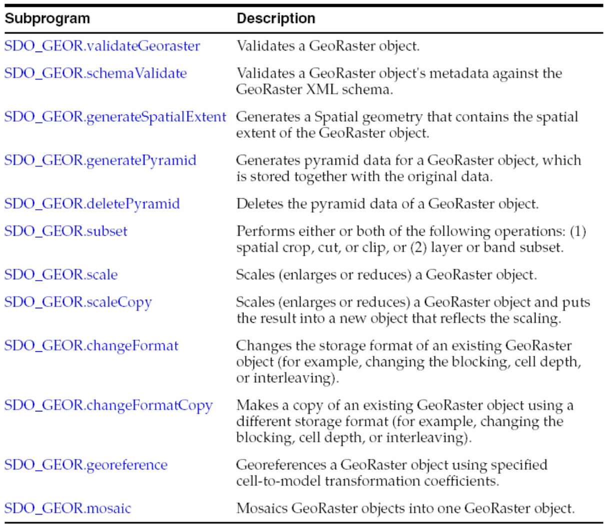


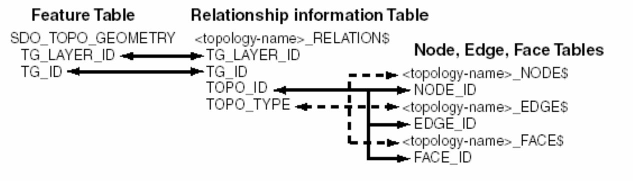
Table 1–3 Subprograms to Validate and Process GeoRaster Objects



Subprogram	Description
SDO_GEOR.validateGeoraster	Validates a GeoRaster object.
SDO_GEOR.schemaValidate	Validates a GeoRaster object's metadata against the GeoRaster XML schema.
SDO_GEOR.generateSpatialExtent	Generates a Spatial geometry that contains the spatial extent of the GeoRaster object.
SDO_GEOR.generatePyramid	Generates pyramid data for a GeoRaster object, which is stored together with the original data.
SDO_GEOR.deletePyramid	Deletes the pyramid data of a GeoRaster object.
SDO_GEOR.subset	Performs either or both of the following operations: (1) spatial crop, cut, or clip, or (2) layer or band subset.
SDO_GEOR.scale	Scales (enlarges or reduces) a GeoRaster object.
SDO_GEOR.scaleCopy	Scales (enlarges or reduces) a GeoRaster object and puts the result into a new object that reflects the scaling.
SDO_GEOR.changeFormat	Changes the storage format of an existing GeoRaster object (for example, changing the blocking, cell depth, or interleaving).
SDO_GEOR.changeFormatCopy	Makes a copy of an existing GeoRaster object using a different storage format (for example, changing the blocking, cell depth, or interleaving).
SDO_GEOR.georeference	Georeferences a GeoRaster object using specified cell-to-model transformation coefficients.
SDO_GEOR.mosaic	Mosaics GeoRaster objects into one GeoRaster object.

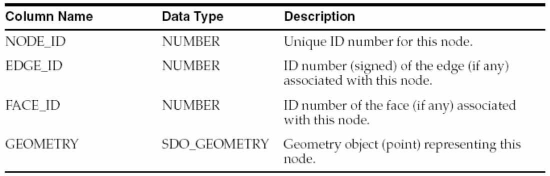
Topology and Network

Figure 1–5 Mapping Between Feature Tables and Topology Tables



Nodes Table

Table 1–3 Columns in the <topology-name>_NODE$ Table



Column Name	Data Type	Description
NODE_ID	NUMBER	Unique ID number for this node.
EDGE_ID	NUMBER	ID number (signed) of the edge (if any) associated with this node.
FACE_ID	NUMBER	ID number of the face (if any) associated with this node.
GEOMETRY	SDO_GEOMETRY	Geometry object (point) representing this node.

Edges table

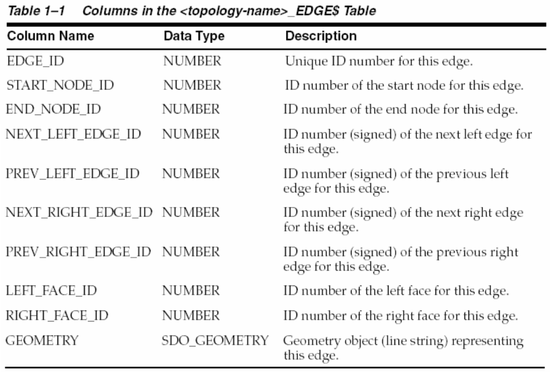
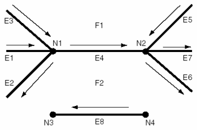


Table 1–1 Columns in the <topology-name>_EDGE$ Table

Column Name	Data Type	Description
EDGE_ID	NUMBER	Unique ID number for this edge.
START_NODE_ID	NUMBER	ID number of the start node for this edge.
END_NODE_ID	NUMBER	ID number of the end node for this edge.
NEXT_LEFT_EDGE_ID	NUMBER	ID number (signed) of the next left edge for this edge.
PREV_LEFT_EDGE_ID	NUMBER	ID number (signed) of the previous left edge for this edge.
NEXT_RIGHT_EDGE_ID	NUMBER	ID number (signed) of the next right edge for this edge.
PREV_RIGHT_EDGE_ID	NUMBER	ID number (signed) of the previous right edge for this edge.
LEFT_FACE_ID	NUMBER	ID number of the left face for this edge.
RIGHT_FACE_ID	NUMBER	ID number of the right face for this edge.
GEOMETRY	SDO_GEOMETRY	Geometry object (line string) representing this edge.



Faces Table

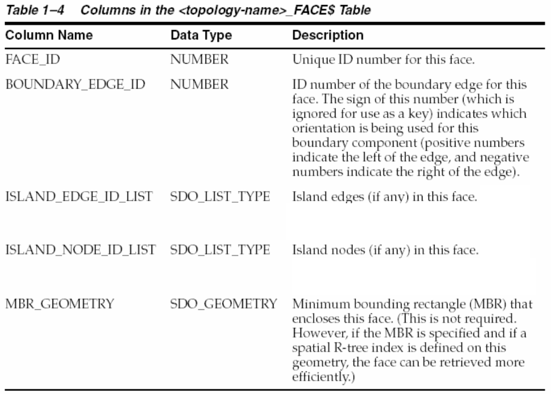


Table 1–4 Columns in the <topology-name>_FACE$ Table

Column Name	Data Type	Description
FACE_ID	NUMBER	Unique ID number for this face.
BOUNDARY_EDGE_ID	NUMBER	ID number of the boundary edge for this face. The sign of this number (which is ignored for use as a key) indicates which orientation is being used for this boundary component (positive numbers indicate the left of the edge, and negative numbers indicate the right of the edge).
ISLAND_EDGE_ID_LIST	SDO_LIST_TYPE	Island edges (if any) in this face.
ISLAND_NODE_ID_LIST	SDO_LIST_TYPE	Island nodes (if any) in this face.
MBR_GEOMETRY	SDO_GEOMETRY	Minimum bounding rectangle (MBR) that encloses this face. (This is not required. However, if the MBR is specified and if a spatial R-tree index is defined on this geometry, the face can be retrieved more efficiently.)

Creating the topology

```
-- Create the topology. (Null SRID in this example.)
EXECUTE SDO_TOPO.CREATE_TOPOLOGY('LAND_USE_HIER', 0.00005);
-- Create feature tables.
CREATE TABLE land_parcels ( -- Land parcels (selected faces)
   feature_name VARCHAR2(30) PRIMARY KEY,
   feature SDO_TOPO_GEOMETRY);
CREATE TABLE block_groups (
   feature_name VARCHAR2(30) PRIMARY KEY,
   feature SDO_TOPO_GEOMETRY);
CREATE TABLE tracts (
   feature_name VARCHAR2(30) PRIMARY KEY,
   feature SDO_TOPO_GEOMETRY);
CREATE TABLE counties (
   feature_name VARCHAR2(30) PRIMARY KEY,
   feature SDO_TOPO_GEOMETRY);
CREATE TABLE states (
   feature_name VARCHAR2(30) PRIMARY KEY,
   feature SDO_TOPO_GEOMETRY);
```

Exemple PL/SQL

```
DECLARE
   land_parcels_id NUMBER;
   block_groups_id NUMBER;
   tracts_id NUMBER;
   counties_id NUMBER;
BEGIN
   SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'LAND_PARCELS',
        'FEATURE','POLYGON');
   SELECT tg_layer_id INTO land_parcels_id FROM user_sdo_topo_info
   WHERE topology = 'LAND_USE_HIER' AND table_name = 'LAND_PARCELS';
   SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'BLOCK_GROUPS',
        'FEATURE','POLYGON', NULL, land_parcels_id);
   SELECT tg_layer_id INTO block_groups_id FROM user_sdo_topo_info
        Topology Data Model Tables
        WHERE topology = 'LAND_USE_HIER' AND table_name = 'BLOCK_GROUPS';
   SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'TRACTS',
        'FEATURE','POLYGON', NULL, block_groups_id);
   SELECT tg_layer_id INTO tracts_id FROM user_sdo_topo_info
        WHERE topology = 'LAND_USE_HIER' AND table_name = 'TRACTS';
   SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'COUNTIES',
        'FEATURE','POLYGON', NULL, tracts_id);
   SELECT tg_layer_id INTO counties_id FROM user_sdo_topo_info
        WHERE topology = 'LAND_USE_HIER' AND table_name = 'COUNTIES';
   SDO_TOPO.ADD_TOPO_GEOMETRY_LAYER('LAND_USE_HIER', 'STATES',
        'FEATURE','POLYGON', NULL, counties_id);
END;
```

SDO_TOPO_GEOMETRY Type

```
CREATE TYPE sdo_topo_geometry AS OBJECT
  (tg_type NUMBER,
   tg_id NUMBER,
   tg_layer_id NUMBER,
   topology_id NUMBER
  );
```

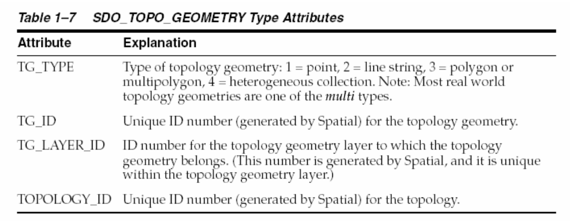
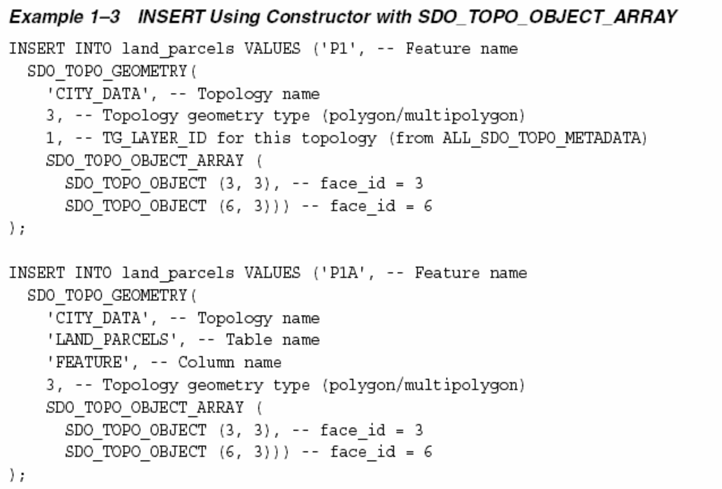


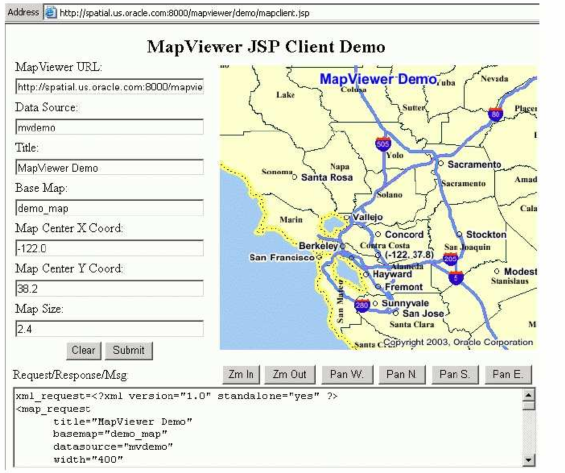
Table 1–7 SDO_TOPO_GEOMETRY Type Attributes

Attribute	Explanation
TG_TYPE	Type of topology geometry: 1 = point, 2 = line string, 3 = polygon or multipolygon, 4 = heterogeneous collection. Note: Most real world topology geometries are one of the *multi* types.
TG_ID	Unique ID number (generated by Spatial) for the topology geometry.
TG_LAYER_ID	ID number for the topology geometry layer to which the topology geometry belongs. (This number is generated by Spatial, and it is unique within the topology geometry layer.)
TOPOLOGY_ID	Unique ID number (generated by Spatial) for the topology.



Example 1–3 INSERT Using Constructor with SDO_TOPO_OBJECT_ARRAY

```
INSERT INTO land_parcels VALUES ('P1', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    3, -- Topology geometry type (polygon/multipolygon)
    1, -- TG_LAYER_ID for this topology (from ALL_SDO_TOPO_METADATA)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 3), -- face_id = 3
      SDO_TOPO_OBJECT (6, 3))) -- face_id = 6
);

INSERT INTO land_parcels VALUES ('P1A', -- Feature name
  SDO_TOPO_GEOMETRY(
    'CITY_DATA', -- Topology name
    'LAND_PARCELS', -- Table name
    'FEATURE', -- Column name
    3, -- Topology geometry type (polygon/multipolygon)
    SDO_TOPO_OBJECT_ARRAY (
      SDO_TOPO_OBJECT (3, 3), -- face_id = 3
      SDO_TOPO_OBJECT (6, 3))) -- face_id = 6
);
```



Address http://spatial.us.oracle.com:8000/mapviewer/demo/mapclient.jsp

MapViewer JSP Client Demo

MapViewer URL: http://spatial.us.oracle.com:8000/mapvie

Data Source: mvdemo

Title: MapViewer Demo

Base Map: demo_map

Map Center X Coord: -122.0

Map Center Y Coord: 38.2

Map Size: 2.4

Request/Response/Msg:

```
xml_request=<?xml version="1.0" standalone="yes" ?>
<map_request
      title="MapViewer Demo"
      basemap="demo_map"
      datasource="mvdemo"
      width="400"
```

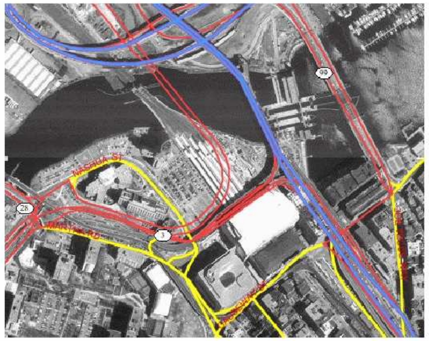
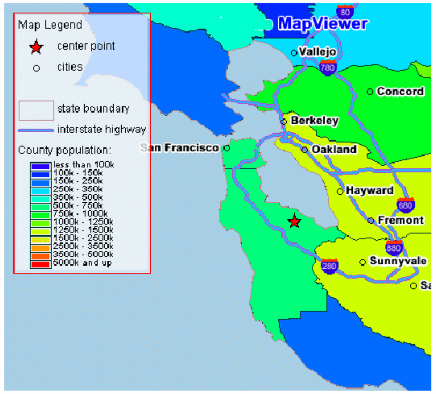


Figure 2–1 Image Theme and Other Themes Showing Boston Roadways

Figure 2–2 Map with Legend



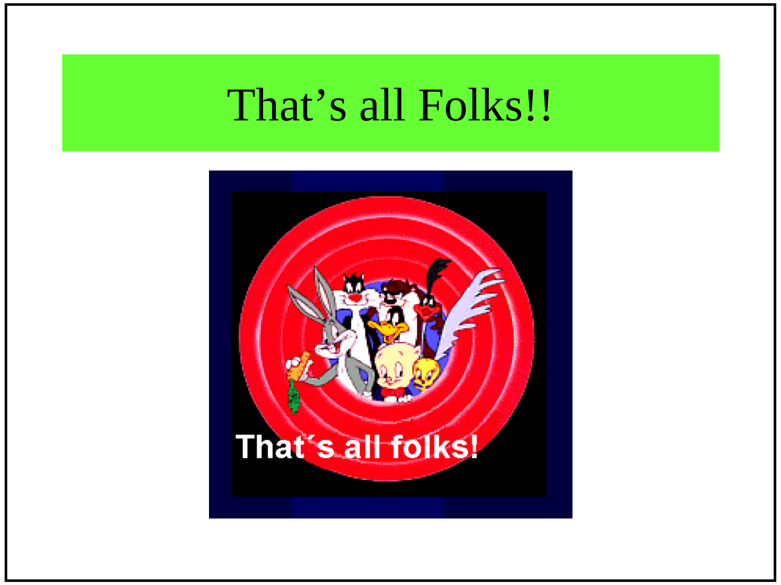
```
declare
    l_http_req utl_http.req;
    l_http_resp utl_http.resp;
    l_url varchar2(4000):= 'http://my_corp.com:8888/mapviewer/omserver';
    l_value varchar2(4000);
    img_url varchar2(4000);
    response sys.xmltype;
    output varchar2(255);
    map_req varchar2(4000);
begin
    utl_http.set_persistent_conn_support(TRUE);
    map_req := '<?xml version="1.0" standalone="yes"?>
    <map_request title="MapViewer Demonstration"
                    datasource="mvdemo"
                    basemap="course_map"
                    Map Request Examples
                    width="500"
                    height="375"
                    bgcolor="#a6cae0"
                    antialiasing="false"
                    format="GIF_URL">
          <center size="5" >
             <geoFeature>
                <geometricProperty>
                   <Point>
                       <coordinates>-122.2615, 37.5266</coordinates>
                   </Point>
                </geometricProperty>
             </geoFeature>
          </center>
    </map_request>';
```

Exemple d'interactions entre PL/SQL et Map Viewer

```
   l_http_req := utl_http.begin_request(l_url, 'POST', 'HTTP/1.0');
   --
   -- sets up proper HTTP headers
   --
   utl_http.set_header(l_http_req, 'Content-Type',
        'application/x-www-form-urlencoded');
   utl_http.set_header(l_http_req, 'Content-Length',
   length('xml_request=' ||   map_req));
   utl_http.set_header(l_http_req, 'Host', 'my_corp.com');
   utl_http.set_header(l_http_req, 'Port', '8888');
   utl_http.write_text(l_http_req, 'xml_request=' || map_req);
   --
   l_http_resp := utl_http.get_response(l_http_req);
   utl_http.read_text(l_http_resp, l_value);
   response := sys.xmltype.createxml (l_value);
   utl_http.end_response(l_http_resp);
   img_url := response.extract('/map_response/map_image/map_
        content/@url').getstringval();
   dbms_output.put_line(img_url);
end;
```

8.5 – Remarques finales

- de "Spatial Data Option" à Oracle 10g
- Traitement des données spatiales
- Intégré dans la plupart des SIG



That’s all Folks!!