

Tutorial Oracle Spatial

Tables des matières

1.	Résumé.....	2
2.	Introduction.....	2
3.	Les concepts de base.....	3
a.	Les types géométriques.....	3
b.	Le modèle de données Spatial.....	4
i.	Elément.....	4
ii.	Objet géométrique.....	4
iii.	Couche.....	4
iv.	Système de coordonnées spatiales.....	4
v.	Tolérance.....	4
4.	Les différentes commandes Spatiales.....	5
a.	Marche à suivre pour créer une base de donnée spatiale.....	5
b.	Tester sa base avec les opérateurs spatiaux.....	7
i.	SDO_FILTER(geometry1, geometry2, {params});.....	7
ii.	SDO_NN(geometry1, geometry2, param [, number]);.....	7
iii.	SDO_NN_DISTANCE(number);.....	7
iv.	SDO_WITHIN_DISTANCE(geometry1, aGeom, params);.....	8
v.	SDO_RELATE(geometry1, geometry2, {params});.....	8
c.	Exemples de commandes simples.....	10
i.	Insérer un polygone.....	10
ii.	La liste des points dont la coordonnée X vaut 12.....	10
iii.	Crée un indexe simple et fixe.....	10
iv.	Créer un filtre primaire.....	10
5.	Conclusions.....	11
6.	Bibliographie.....	12
7.	Annexes.....	13
a.	Liste des commandes tirées de la documentation oracle.....	13
b.	Exemples complets de codes Spatiales.....	18
i.	Exemple d'insertion, d'indexation et de requêtes sur une base de données Spatale.....	18

1. Résumé

Le présent document est un tutorial pour la gestion des données spatiales des bases de données Oracle. Il n'est de loin pas exhaustif, mais donne un aperçu des possibilités offertes par Oracle spatial et ces concurrents. Pour une information optimale, je vous conseille de vous référer à la documentation d'oracle.

2. Introduction

Introduit avec « Oracle 8 », « Oracle Spatial » ajoute une 3^{ème} dimension à la gestion déjà existante des coordonnées dans le plan « Oracle Locator ».

Pour celles et ceux qui se demandent ce qu'est une donnée spatiale, voici un exemple : Une carte routière est un objet bidimensionnel qui contient des points, des traits et des polygones qui représentent des localités, des routes/chemins de fer et des régions/pays. Une carte est une représentation d'informations géographiques. Il s'agit d'une projection des données réelles des localités routes et régions qui a pour but de visualiser les distances relatives et les coordonnées de ces données. La donnée qui permet de donner les coordonnées et l'altitude est une donnée spatiale.

« Oracle Spatial » permet une gestion simple et efficace de ce type de données. Il s'agit donc d'un outil essentiel à toute gestion d'informations spatiales, que ce soit dans les domaines aussi diversifiés que la téléphonie mobile, le Guidage Par Satellites, les sites de cartographies, les systèmes d'informations géographiques, architectures etc...

Remarque : Si vous désirez essayer les codes, il faut utiliser un client « sqlplus » ou « isqlplus » et bien entendu un serveur configuré de la bonne manière. Si je n'ai rien écrit à ce sujet c'est qu'il me manque déjà beaucoup de page pour décrire les fondamentaux d'Oracle spatial.

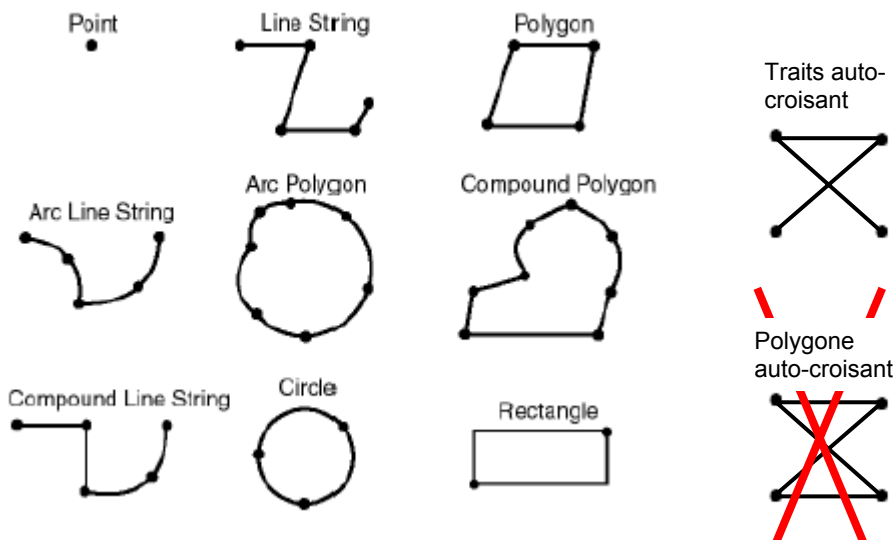
3. Les concepts de base

« Oracle Spatial » est une librairie de fonctions et procédures qui permettent de classer, d'accéder et d'analyser rapidement une base de données Oracle.

Les données spatiales représentent les caractéristiques des emplacements d'objets réels ou conceptuels dans un environnement réels ou conceptuels dans lequel ils existent.

a. Les types géométriques

Les figures géométriques sont des séquences de sommets qui sont reliées entre elles par des traits droits ou courbes. La sémantique de ces figures géométrique est déterminée par son type. Voici la liste des types de figures géométriques bidimensionnelles gérées par « Oracle Spatial ».



Remarque : l'énumération des sommets se fait dans le sens des aiguilles d'une montre.

Attention à ne pas créer de polygone dont les traits se croisent car cela produit une erreur ! En effet une telle figure à deux surfaces à calculer. Par contre les traits peuvent se croiser car il n'y a pas de surface à calculer.

Important :

« Oracle Spatial » permet aussi de gérer les figures tri et quadri-dimensionnelles, mais les fonctions spatiales ne traitent que les deux premières dimensions (sauf pour les fonctions MBR et LRS) et les filtres sont désactivés si un indice a été créé sur plus de deux dimensions.

b. Le modèle de données Spatial

Le modèle de données Spatial est une structure hiérarchique composée d'éléments, d'objets géométriques et de couches. Où les couches sont composés d'objets géométriques, eux-même composés d'éléments.

i. Élément

Un élément une variable de type géométrique (voir 3.a).

ii. Objet géométrique

Est composé de un ou plusieurs éléments qui ne sont pas forcément du même type.

iii. Couche

Est composés des objets géométriques ayant des caractéristiques communes. Ex une couche regroupe les objets géométriques concernant la densité de population, alors qu'une autre couche regroupe les objets géométriques concernant la topographie.

iv. Système de coordonnées spatiales

Définit le système de coordonnées spatiales. Le système de coordonnées spatiales peut soit être géo-référencé ou pas et détermine l'unité des coordonnées des points. Si le système n'est pas géo-référencé, il est cartésien, ce qui veut dire qu'il n'est pas apparenté a une représentation de la terre.

v. Tolérance

Définir la précision des données spatiales, cela permet donc de savoir par exemple quel est l'écart maximum entre deux points qui sont considérés comme communs. Si le système de coordonnées est métrique, une tolérance de 100 veut dire une tolérance de 100 mètres.

4. Les différentes commandes Spatiales

Vous présenter la totalité des commandes n'a pas un grand intérêt, je cite juste certaines à titre d'exemple. La totalité des commandes se trouvent dans les annexes, mais il ne s'agit que d'une énumération et non d'une spécification. La spécification des commandes SQL se trouve dans la documentation d'oracle dont je recommande vivement la lecture.

a. Marche à suivre pour créer une base de donnée spatiale

1. Créer la/les tables pour enregistrer les données spatiales.

Ex :

```
CREATE TABLE nom_table (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  'forme' MDSYS.SDO_GEOMETRY
);
```

2. Insérer les enregistrements dans les tables.

Ex :

```
INSERT INTO nom_table VALUES(
  1,
  'nom_zone',
  MDSYS.SDO_GEOMETRY(
    2003, -- polygone à 2 dimensions
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), -- un rectangle (1003 = exterior)
    MDSYS.SDO_ORDINATE_ARRAY(1,1, 5,7) -- 2 coordonnées utiles pour un
    rectangle
    -- coin inférieur gauche et supérieur droit | format des données est cartésien
    -- il ne représente rien de réel.
  )
);
```

3. Mettre à jour les méta-données de la vue nécessaire pour pouvoir créer les indexes à faire une fois pour chaque couche

Ex :

```
INSERT INTO USER_SDO_GEOM_METADATA VALUES (
  'nom_table',
  'forme',
  MDSYS.SDO_DIM_ARRAY( -- 20X20 grid
    MDSYS.SDO_DIM_ELEMENT('X', 0, 20, 0.005), -- X
    MDSYS.SDO_DIM_ELEMENT('Y', 0, 20, 0.005) -- Y
  ),
  NULL -- SRID
);
```

4. Créer un index spatial sur au moins une des colonnes de chaque table.

Ex :

```
CREATE INDEX <nom_index>  
ON nom_table (forme)  
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

b. Tester sa base avec les opérateurs spatiaux

i. SDO_FILTER(geometry1, geometry2, {params});

Premier filtre. Détermine si les objets géométriques peuvent interagir.

geometry1 et 2:

Colonne de type MDSYS.SDO_GEOMETRY de la table sur lesquels un index doit être défini.

params

querytype : **Obligatoire** soit WINDOW, soit JOIN.

WINDOW : **Conseillé**, test seulement les objets qui peuvent interagir.

JOIN : **Déconseillé**, test tous les objets.

idxtab1 : Index de geometry1 à utiliser. A définir que si geometry1 a plusieurs indexes.

idxtab2 : Index de geometry2 à utiliser. A définir que si geometry2 a plusieurs indexes.

Ex :

```
SELECT A.gid
FROM Polygons A, query_polys B
WHERE B.gid = 1
AND SDO_FILTER(A.Geometry, B.Geometry, 'querytype = WINDOW') = 'TRUE';
```

ii. SDO_NN(geometry1, geometry2, param [, number]);

Retourne le voisin le plus proche.

geometry1 et 2 :

Colonne de type MDSYS.SDO_GEOMETRY de la table sur lesquels un index doit être défini.

param : **mots clés** sdo_batch_size, sdo_num_res et unit

sdo_batch_size : Nombre d'évaluations simultanées (utilisable si index de type R-tree).

sdo_num_res : Spécifie le nombre de résultats devant être retourné.

Unit : Définit l'unité du système de coordonnées spatial.

Number : Si SDO_NN_DISTANCE est utilisé, alors les deux number doivent être égaux.

Ex :

```
SELECT r.name FROM restaurants r WHERE
SDO_NN(r.geometry, :my_hotel, 'sdo_batch_size=10') = 'TRUE'
AND r.cuisine = 'Italian' AND ROWNUM <=2;
```

iii. SDO_NN_DISTANCE(number);

Retourne la distance avec le voisin le plus proche retourné par SDO_NN (pas utilisable seul).

Number : Doit être égal au number de SDO_NN.

Ex :

```
SELECT /*+ INDEX(coła_markets coła_spatial_idx) */
c.mkt_id, c.name, mdsys.SDO_NN_DISTANCE(1) dist
FROM coła_markets c
WHERE SDO_NN(c.shape, mdsys.sdo_geometry(2001, NULL,
mdsys.sdo_point_type(10,7,NULL), NULL, NULL),
'sdo_num_res=2', 1) = 'TRUE' ORDER BY dist;
```

iv. **SDO_WITHIN_DISTANCE(geometry1, aGeom, params);**

Détermine si deux éléments/objets géométriques sont à une distance inférieure à « distance » l'un de l'autre.

geometry1 :

Colonne de type MDSYS.SDO_GEOMETRY de la table sur lesquels un index doit être défini.

aGeom :

Spécifie l'objet à tester.

params

distance : définit la distance.

idxtab1 : Index de geometry1 à utiliser. A définir que si geometry1 a plusieurs indexes.

querytype : si « querytype=FILTER » n'utilise que le premier filtre, sinon utilise les 2.

unit : Définit l'unité du système de coordonnées spatial (pour la distance).

Ex :

```
SELECT A.GID
FROM POLYGONS A
WHERE
SDO_WITHIN_DISTANCE(A.Geometry, :aGeom, 'distance = 10') = 'TRUE';
```

v. **SDO_RELATE(geometry1, geometry2, {params});**

Second filtre, détermine les différentes relations possibles entre les éléments/objets géométriques qui fonctionne de la manière suivante :

Boundary(l'extérieur) : polygone qui sépare l'élément/objet géométrique du reste du système de coordonnées.

Interior(l'intérieur) : polygone se trouvant à l'intérieur de l'extérieur ☺.

geometry1 et 2:

Colonne de type MDSYS.SDO_GEOMETRY de la table sur lesquels un index doit être défini.

params

querytype : **Obligatoire** soit WINDOW, soit JOIN.

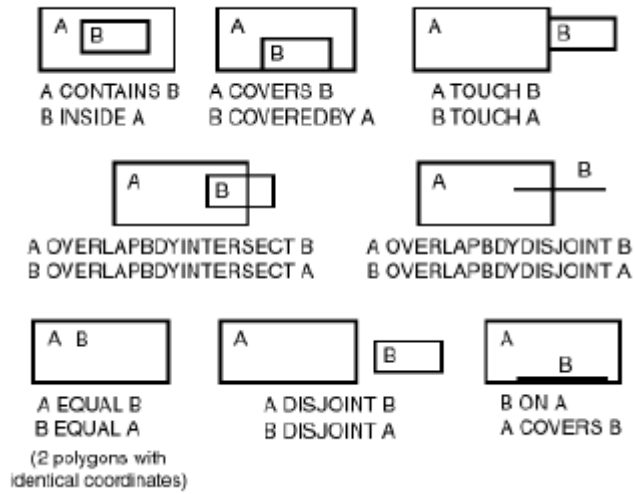
WINDOW : **Conseillé**, test seulement les objets qui peuvent interagir.

JOIN : **Déconseillé**, test tous les objets.

idxtab1 : Index de geometry1 à utiliser. A définir que si geometry1 a plusieurs indexes.

I :dxtab2 : Index de geometry2 à utiliser. A définir que si geometry2 a plusieurs indexes.

mask : doit être égal à l'un ou à la somme des mots clés suivants.



Ex :

```

SELECT a.gid
FROM polygons a, query_polys B
WHERE B.gid = 1
  AND SDO_RELATE(A.Geometry, B.Geometry,
    'mask=inside querytype=WINDOW') = 'TRUE'
UNION ALL
SELECT a.gid
FROM polygons a, query_polys B
WHERE B.gid = 1
  AND SDO_RELATE(A.Geometry, B.Geometry,
    'mask=coveredby querytype=WINDOW') = 'TRUE';
ou alors
'mask=inside + coveredby' querytype=WINDOW') = 'TRUE';

```

c. Exemples de commandes simples

i. Insérer un polygone

```
INSERT INTO nom_table VALUES(
  2,
  nom_zone,
  MDSYS.SDO_GEOMETRY( -- Polygone à 4 côtés
    2003,
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1), -- exterior = 1003
    MDSYS.SDO_ORDINATE_ARRAY(5,1, 8,1, 8,6, 5,7, 5,1) - coordonnées cartésiennes
  )
);
```

ii. La liste des points dont la coordonnée X vaut 12

```
SELECT * from cola_markets c WHERE c.shape.SDO_POINT.X = 12;
```

iii. Créée un indexe simple et fixe

```
CREATE INDEX ROADS_FIXED ON ROADS(SHAPE) INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('SDO_LEVEL=8');
```

iv. Créer un filtre primaire

```
SELECT A.Feature_ID FROM TARGET A, WINDOWS B
WHERE B.ID = 'WINS_1'
AND sdo_filter(A.shape, B.shape, 'querytype=window') = 'TRUE';
```

5. Conclusions

« Oracle Spatial » est un outils très complet qu'il n'est pas aisé de décrire en quelque pages. Ce que je me suis efforcé de faire malgré tout, Ce tutorial permet de créer une première base de données Spatiale, tout en permettant de se faire une petite idée du travail du moteur de recherche.

Mais à mon avis, ce n'est pas suffisant, il est nettement plus intéressant de lire la documentation mise à disposition sur le site.

6. Bibliographie

Technologies Oracle :
<http://www.otn.oracle.com>

7. Annexes

a. Liste des commandes tirées de la documentation oracle

Table 8–1 Spatial Index Creation and Usage Statements

Statement	Description
ALTER INDEX	Alters a spatial index on a column of type MDSYS.SDO_GEOMETRY.
ALTER INDEX REBUILD	Rebuilds a spatial index on a column of type MDSYS.SDO_GEOMETRY.
ALTER INDEX RENAME TO	Changes the name of a spatial index on a column of type MDSYS.SDO_GEOMETRY.
CREATE INDEX	Creates a spatial index on a column of type MDSYS.SDO_GEOMETRY.
DROP INDEX	Deletes a spatial index on a column of type MDSYS.SDO_GEOMETRY.

Table 9–1 SDO_GEOMETRY Type Methods

Method	Description
GET_DIMS	Returns the number of dimensions of a geometry object.
GET_GTYPE	Returns the geometry type of a geometry object.
GET_LRS_DIM	Returns the measure dimension of an LRS geometry object.

Table 10–1 Spatial Usage Operators

Operator	Description
SDO_FILTER	Specifies which geometries may interact with a given geometry.
SDO_NN	Determines the nearest neighbor geometries to a geometry.
SDO_NN_DISTANCE	Returns the distance of an object returned by the SDO_NN operator.
SDO_RELATE	Determines whether or not two geometries interact in a specified way.
SDO_WITHIN_DISTANCE	Determines if two geometries are within a specified distance from one another.

Table 11–1 Geometry Functions

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.

Table 11–1 Geometry Functions (Cont.)

Function	Description
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 (dimension) 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 (dimension) 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 distance from one another.

Table 12–1 Spatial Aggregate Functions

Method	Description
SDO_AGGR_CENTROID	Returns a geometry object that is the centroid ("center of gravity") of the specified geometry objects.
SDO_AGGR_CONVEXHULL	Returns a geometry object that is the convex hull of the specified geometry objects.
SDO_AGGR_LRS_CONCAT	Returns an LRS geometry object that concatenates specified LRS geometry objects.
SDO_AGGR_MBR	Returns the minimum bounding rectangle of the specified geometry objects
SDO_AGGR_UNION	Returns a geometry object that is the topological union (<i>OR</i> operation) of the specified geometry objects.

Table 13–1 Functions and Procedures for Coordinate Systems

Function	Description
SDO_CS.TRANSFORM	Transforms a geometry representation using a coordinate system (specified by SRID or name).
SDO_CS.TRANSFORM_LAYER	Transforms an entire layer of geometries (that is, all geometries in a specified column in a table).
SDO_CS.VIEWPORT_TRANSFORM	Transforms an optimized rectangle into a valid polygon for use with Spatial operators and functions.

Table 14–1 Functions for Creating and Editing Geometric Segments

Function	Description
SDO_LRS.DEFINE_GEOM_SEGMENT (procedure)	Defines a geometric segment.
SDO_LRS.REDEFINE_GEOM_SEGMENT (procedure)	Populates the measures of all shape points of a geometric segment based on the start and end measures, overriding any previously assigned measures between the start point and end point.
SDO_LRS.CLIP_GEOM_SEGMENT	Clips a geometric segment (synonym of SDO_LRS.DYNAMIC_SEGMENT).
SDO_LRS.DYNAMIC_SEGMENT	Clips a geometric segment (synonym of SDO_LRS.CLIP_GEOM_SEGMENT).

Table 14–1 Functions for Creating and Editing Geometric Segments (Cont.)

Function	Description
SDO_LRS.CONCATENATE_GEOM_SEGMENTS	Concatenates two geometric segments into one segment.
SDO_LRS.OFFSET_GEOM_SEGMENT	Returns the geometric segment at a specified offset from a geometric segment.
SDO_LRS.SCALE_GEOM_SEGMENT	Scales a geometric segment.
SDO_LRS.SPLIT_GEOM_SEGMENT (procedure)	Splits a geometric segment into two segments.
SDO_LRS.REVERSE_MEASURE	Returns a new geometric segment by reversing the original geometric segment.
SDO_LRS.TRANSLATE_MEASURE	Returns a new geometric segment by translating the original geometric segment (that is, shifting the start and end measures by a specified value).
SDO_LRS.REVERSE_GEOMETRY	Returns a new geometric segment by reversing the measure values and the direction of the original geometric segment.

Table 14–2 lists functions related to querying geometric segments.

Table 14–2 Functions for Querying Geometric Segments

Function	Description
SDO_LRS.VALID_GEOM_SEGMENT	Checks if a geometric segment is valid.
SDO_LRS.VALID_LRS_PT	Checks if an LRS point is valid.
SDO_LRS.VALID_MEASURE	Checks if a measure falls within the measure range of a geometric segment.
SDO_LRS.CONNECTED_GEOM_SEGMENTS	Checks if two geometric segments are connected.
SDO_LRS.GEOM_SEGMENT_LENGTH	Returns the length of a geometric segment.
SDO_LRS.GEOM_SEGMENT_START_PT	Returns the start point of a geometric segment.
SDO_LRS.GEOM_SEGMENT_END_PT	Returns the end point of a geometric segment.
SDO_LRS.GEOM_SEGMENT_START_MEASURE	Returns the start measure of a geometric segment.

Table 15–1 Migration Procedures

Procedure	Description
SDO_MIGRATE.FROM_815_TO_81X	Migrates data from Spatial release 8.1.5 to the current release.
SDO_MIGRATE.OGIS_METADATA_FROM	Generates a temporary table used when migrating OGIS (OpenGIS) metadata tables.
SDO_MIGRATE.OGIS_METADATA_TO	Reads a temporary table used when migrating OGIS metadata tables.
SDO_MIGRATE.TO_734	Migrates data from a previous release of Spatial Data Option to release 7.3.4.
SDO_MIGRATE.TO_81X	Migrates tables from Spatial Data Option release 7.3.4 or Spatial Cartridge release 8.0.4 to Oracle Spatial.
SDO_MIGRATE.TO_CURRENT	Migrates data from a previous Spatial release to the current release.

Table 16–1 Tuning Functions and Procedures

Function/Procedure	Description
SDO_TUNE.ANALYZE_RTREE	Analyzes an R-tree index; generates statistics about the index use, and recommends a rebuild of the index if a rebuild would improve query performance significantly.
SDO_TUNE.AVERAGE_MBR	Calculates the average minimum bounding rectangle for geometries in a layer.
SDO_TUNE.ESTIMATE_INDEX_PERFORMANCE	Estimates the spatial index selectivity.
SDO_TUNE.ESTIMATE_TILING_LEVEL	Determines an appropriate tiling level for creating fixed-size index tiles.
SDO_TUNE.ESTIMATE_TILING_TIME	Estimates the tiling time for a layer, in seconds.
SDO_TUNE.ESTIMATE_TOTAL_NUMTILES	Estimates the total number of spatial tiles for a layer.
SDO_TUNE.EXTENT_OF	Determines the minimum bounding rectangle of the data in a layer.
SDO_TUNE.HISTOGRAM_ANALYSIS	Calculates statistical histograms for a spatial layer.
SDO_TUNE.MIX_INFO	Calculates geometry type information for a spatial layer, such as the percentage of each geometry type.

Table 16–1 Tuning Functions and Procedures

Function/Procedure	Description
SDO_TUNE.QUALITY_DEGRADATION	Returns the quality degradation for an R-tree index or the average quality degradation for all index tables for an R-tree index.
SDO_TUNE.RTREE_QUALITY	Returns the quality score for an R-tree index or the average quality score for all index tables for an R-tree index.

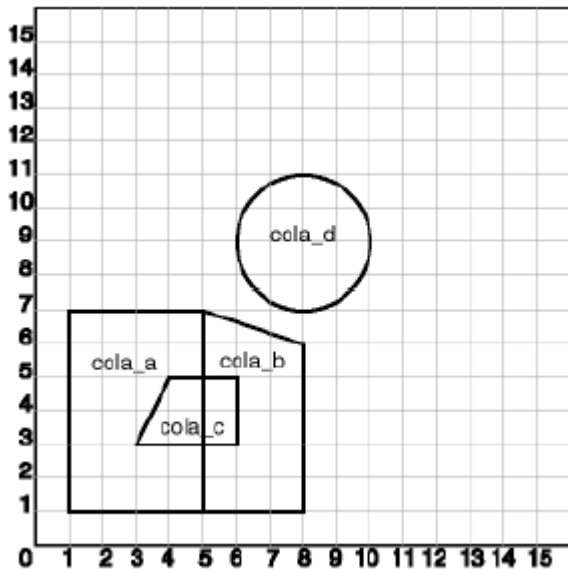
Table 17–1 Utility Functions and Procedures

Function/Procedure	Description
SDO_UTIL.EXTRACT	Returns the geometry that represents a specified element (and optionally a ring) of the input geometry.
SDO_UTIL.GETVERTICES	Returns the coordinates of the vertices of the input geometry.

b. Exemples complets de codes Spatiales

i. Exemple d'insertion, d'indexation et de requêtes sur une base de données Spatiale

Modèle crée :



Marche à suivre :

- Créer une table (COLA_Markets) pour enregistrer les données spatiales.
- Insérer les 4 zones d'intérêts (cola_a, cola_b, cola_c, cola_d).
- Mise à jour de la méta-donnée de la vue.
- Créer un indexe spatial (COLA_SPATIAL_IDX).
- Quelques requêtes pour tester cette table.

```
-- Create a table for cola (soft drink) markets in a
-- given geography (such as city or state).
-- Each row will be an area of interest for a specific
-- cola (for example, where the cola is most preferred
-- by residents, where the manufacturer believes the
-- cola has growth potential, and so on).
-- (For restrictions on spatial table and column names, see
-- Section 2.4.1 and Section 2.4.2.)
CREATE TABLE cola_markets (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape MDSYS.SDO_GEOMETRY
);

-- The next INSERT statement creates an area of interest for
-- Cola A. This area happens to be a rectangle.
-- The area could represent any user-defined criterion: for
-- example, where Cola A is the preferred drink, where
-- Cola A is under competitive pressure, where Cola A
-- has strong growth potential, and so on.
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
)
);

-- The next two INSERT statements create areas of interest for
-- Cola B and Cola C. These areas are simple polygons (but not
-- rectangles).
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)
  )
);

-- Now insert an area of interest for Cola D. This is a
-- circle with a radius of 2. It is completely outside the
-- first three areas of interest.
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)
  )
);

-----
-- UPDATE METADATA VIEW --
-----

-- Update the USER_SDO_GEOM_METADATA view. This is required
-- before the Spatial index can be created. Do this only once for each
-- layer (that is, table-column combination; here: COLA_MARKETS and SHAPE).
INSERT INTO USER_SDO_GEOM_METADATA VALUES (
  'cola_markets',
  'shape',
  MDSYS.SDO_DIM_ARRAY( -- 20X20 grid
    MDSYS.SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    MDSYS.SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
  ),
  NULL -- SRID
);

-----
-- 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');

-----
-- 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';

-- Is a geometry valid?
SELECT c.name, SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(c.shape, 0.005)
FROM cola_markets c WHERE c.name = 'cola_c';

-- Is a layer valid? (First, create the results table.)
CREATE TABLE val_results (sdo_rowid ROWID, result VARCHAR2(2000));
EXECUTE SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT('COLA_MARKETS', 'SHAPE',
'VAL_RESULTS', 2);
SELECT * from val_results;
```