Oracle Spatial/Locator

Spatial databases can store, manage/interpret, and query spatial data. They are important for geographical information systems (GISs) and CAD/CAM systems.

Oracle database supports spatial data by its Locator and Spatial components.

Spatial Data in Oracle

- `SDO_Geometry` and `ST_Geometry` data-types of columns to store **geometries** (points, line-strings, polygons) and their meta-data (a spatial reference system, axes, accuracy, etc.).
- `SDO_GeoRaster` data-type of a column and `SDO_Raster` tables to store geographical data alligned into cells of **a raster**.
- `SDO_Topo` package to create tables `*_Node$`, `*_Edge$`, and `*_Face$` to described **topological layers** described by their nodes, edges, and faces.
- `Node_ID` and `Link_ID` columns to describe **graphs** by their nodes and edges and to apply graph algorithms.

SDO_Geometry Data-type

For geometries, Oracle provides SDO_Geometry [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-datatypes-metadata.html] object data-type.

```
DESCRIBE SDO_Geometry;
```

```
Name          Null? Type
------------- ----- ---------------------------
SDO_GTYPE           NUMBER
SDO_SRID            NUMBER
```

```
SDO_POINT           MDSYS.SDO_POINT_TYPE()
SDO_ELEM_INFO       MDSYS.SDO_ELEM_INFO_ARRAY()
SDO_ORDINATES       MDSYS.SDO_ORDINATE_ARRAY()

METHOD
------
...
```

An example from Oracle Geometry Examples [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-datatypes-metadata.html#GUID-539FDF31-CF9B-4D16-8C5C-B9D4CFD2A891]:

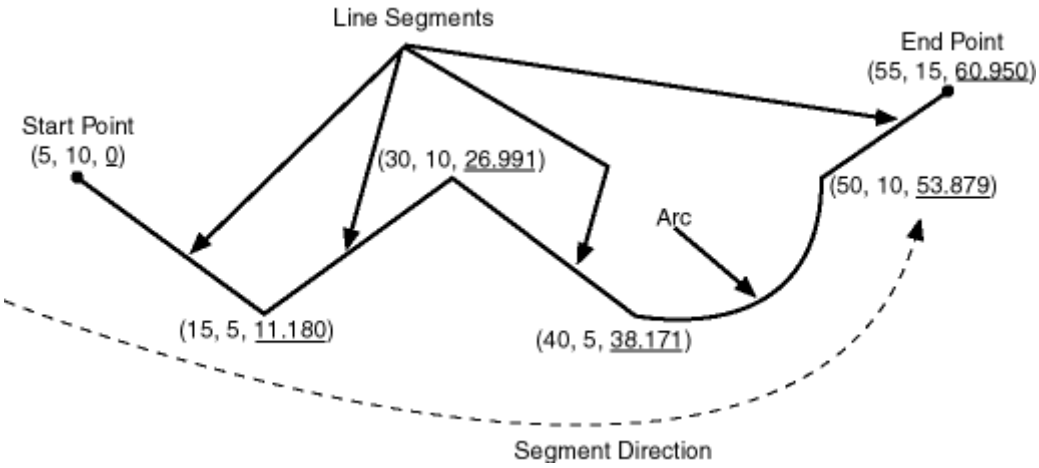
```
CREATE TABLE cola_markets (
  mkt_id NUMBER PRIMARY KEY,
  name VARCHAR2(32),
  shape SDO_GEOMETRY);

INSERT INTO cola_markets VALUES(
  1, 'cola_a',
  SDO_GEOMETRY(
    2003,  -- two-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 =
exterior)
    SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
          -- define rectangle (lower left, i.e., closer to [0,0],
          -- and upper right, i.e. the oposite) with Cartesian-
coordinate data
  )
);
```

The Type of a Geometry (SDO_GType)

`SDO_GTYPE` attribute indicates the type of the geometry as 4 digits in the format DLTT, where:

- `D` identifies the **number of dimensions** (2, 3, or 4)
- `L` identifies the **linear referencing measure dimension** for a three-dimensional linear referencing system (LRS) geometry, that is, which dimension (3 or 4) contains the measure value. For a non-LRS geometry,

specify 0.



- `TT` identifies the **geometry type** (00 through 09, with 10 through 99 reserved for future use):
  - `00` (UNKNOWN_GEOMETRY): Spatial and Graph ignores this geometry.
  - `01` (POINT): Geometry contains one point.
  - `02` (LINE or CURVE): Geometry contains one line string that can contain straight or circular arc segments, or both.
  - `03` (POLYGON or SURFACE): Geometry contains one polygon with or without holes, or one surface consisting of one or more polygons. In a three-dimensional polygon, all points must be on the same plane.
  - `04` (COLLECTION): Geometry is a heterogeneous collection of elements.
  - `05` (MULTIPOINT): Geometry has one or more points.
  - `06` (MULTILINE or MULTICURVE): Geometry has one or more line strings.
  - `07` (MULTIPOLYGON or MULTISURFACE): Geometry can have multiple, disjoint polygons (more than one exterior boundary) or surfaces.
  - `08` (SOLID): Geometry consists of multiple surfaces and is completely enclosed in a three-dimensional space. Can be a cuboid or a frustum.
  - `09` (MULTISOLID): Geometry can have multiple, disjoint solids (more than one exterior boundary).

Spatial Reference System (SDO_SRID)

`SDO_SRID` attribute can be used to identify a coordinate system [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/coordinate-systems-concepts.html] (spatial reference system) to be associated with the geometry.

- If `SDO_SRID` is **null**, the geometry is in Cartesian (local) coordinates.
- If `SDO_SRID` is **not null**, it must contain a value from the `SRID` column of the `SDO_COORD_REF_SYS` table, and this value must be inserted into the `SRID` column of the `USER_SDO_GEOM_METADATA` geometry metadata view [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-datatypes-metadata.html#GUID-AEA8FA01-880B-4865-8074-3B6FBD17A5E1].

An example from Oracle Geometry Examples [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-datatypes-metadata.html#GUID-539FDF31-CF9B-4D16-8C5C-B9D4CFD2A891]:

```
INSERT INTO user_sdo_geom_metadata
    (TABLE_NAME,
     COLUMN_NAME,
     DIMINFO,
     SRID)
  VALUES (
  'cola_markets',
  'shape',
  SDO_DIM_ARRAY(   -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
     ),
  NULL   -- SRID
);
```

The not-null `SDO_SRID` can refer to

- a system of **geodetic coordinates** (geographic coordinates) that are angular coordinates (longitude and latitude), closely related to spherical polar coordinates, and are defined relative to a particular Earth geodetic datum.
- a system of **projected coordinates** that are planar Cartesian coordinates given from performing a mathematical mapping from a point on the Earth's surface to a plane.

Coordinates for a Point Geometry (SDO_Point)

`SDO_POINT` attribute is defined using the `SDO_POINT_TYPE` object type, which has the attributes X, Y, and Z, all of type `NUMBER`.

If the `SDO_ELEM_INFO` and `SDO_ORDINATES` arrays are both null, and the `SDO_POINT` attribute is non-null, then the X, Y, and Z values are considered to be the coordinates for a point geometry. Otherwise, the SDO_POINT attribute is ignored.

Interpretation of Ordinates (SDO_Elem_Info)

`SDO_ELEM_INFO` attribute is defined using a varying length array of numbers to set the interpretation of the ordinates stored in the `SDO_ORDINATES` attribute.

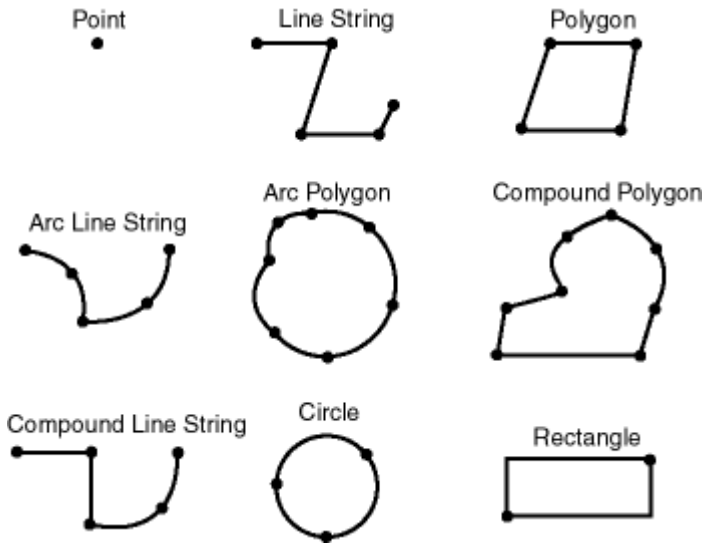
It is the set of triplets `(o1,t1,i1,...,on,tn,in)` where each triplet of numbers is interpreted as follows:

- `o` (SDO_STARTING_OFFSET): Indicates **the offset** within the `SDO_ORDINATES` array where the first ordinate for this element is stored (values start at 1 and not at 0).
- `t` (SDO_ETYPE): Indicates **the type** of the element
  - values 1 (point), 2 (line-string), 1003 (exterior polygon ring), and 2003 (interior polygon ring, i.e., a hole) are considered simple elements,
  - values 4 (compound line-string), 1005 (exterion compound polygon ring), 2005 (interior polygon ring, i.e., a hole), 1006 (exterior surface consisting of one or more polygon rings), and 2006 (interior surface in a solid element, i.e., a hole) are considered compound elements.
- `i` (SDO_INTERPRETATION): For a compound element, it specifies how many subsequent triplet values are part of the element, and for not a compound element, it determines **how the sequence of ordinates is interpreted**:
  - for a point (`t` is 1), 0 is a vector (an oriented point) and 1 is a common point
  - for a line-string (`t` is 2), the intepretation set the line-string's vertices as connected by straight line segments (1), by circular arcs (2), or by

NURBS (non-uniform rational B-spline) curves (3).

- for a simple ploygon (`t` is 1003 or 2003), vertices of the polygin can be connected by straight line segments (1), by circular arcs (2), or the polygon can be an optimized rectangle (3; given by its lower-left corner and the upper-right corner, i.e., the corner closer to [0,0] and its oposite corner, respectively) or a circle (4; given by three distinct non-colinear points, all on the circumference of the circle).

Optimized rectangles and circles cannot be used in geodetic coordinates.



Coordinate Values of the Spatial Object Boundary (SDO_Ordinates)

`SDO_ORDINATES` attribute is defined using a varying length array of `NUMBER` type that stores the coordinate values that make up the boundary of a spatial object. This array must always be used in conjunction with the `SDO_ELEM_INFO` varying length array.

The start and end points for the sequence describing coordinates of a specific element are determined by the `o` (STARTING_OFFSET) values for that element and the next element in the `SDO_ELEM_INFO` array above.

Coordinates of boundaries of exterior polygon rings must be specified in counterclockwise order, and the coordinates of interior polygon rings must be specified in clockwise order.

Validity of Geometries

Breaking rules for the coordinates in `SDO_ORDINATES` according to their interpretation in `SDO_ELEM_INFO` will result into invalid geometries. The validity of geometries can be checked by calling

- `ST_IsValid` method of the `SDO_Geometry` object which returns 0 if a geometry object is invalid or 1 if it is valid with 0.001 as the tolerance value,
- `VALIDATE_GEOMETRY_WITH_CONTEXT` stored function on the `SDO_Geometry` object which returns `TRUE` if a geometry object is valid or an Oracle error message number [http://www.ora-code.com/] based on the specific reason (or `FALSE`) if a geometry is invalid.

```
-- check the validity (there can be an error message with an error
code)
-- with the custom accurance
SELECT name, SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT(geometry, 0.1)
valid -- 0.1 is accurance
FROM city;
-- without the custom accurance (with respect to the accurance set
in the SDO_GEOM_METADATA
SELECT c.name, c.geometry.ST_IsValid()
FROM city c;
```

Spatial Index

The spatial index allows to optimize spatial queries by utilizing spatial operators (see below).

All geometries in a geometry column must have the same `SDO_SRID` value if a spatial index will be built on that column.

An example from Oracle Geometry Examples [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-datatypes-metadata.html#GUID-539FDF31-CF9B-4D16-8C5C-B9D4CFD2A891]:

```
CREATE INDEX cola_spatial_idx
   ON cola_markets(shape)
   INDEXTYPE IS MDSYS.SPATIAL_INDEX_V2;
```

Note: The spatial index version 2 was introduced in Oracle 12cR2 and its main benefit is simplified spatial index management, especially in cases of partitioning.

In the case of indexing of LRS Data, youu must also specify `PARAMETERS('sdo_indx_dims=n')` in the `CREATE INDEX` statement to ensure that only the first `n` dimensions are indexed (e.g., for 3-dimensional geometries with LRS in the fourth dimension, only the first three dimensions should be indexed, i.e., there will be `PARAMETERS('sdo_indx_dims=3')`).

Geometry Examples

For `SDO_Geometry` examples, see Oracle Geometry Examples [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-datatypes-metadata.html#GUID-539FDF31-CF9B-4D16-8C5C-B9D4CFD2A891] and an example of geographical areas of marketing interest [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-datatypes-metadata.html#GUID-51A767CA-82EE-4475-8C8B-13E685C04934].

Querying Spatial Data

Spatial operators provide optimum performance because they use the spatial index for a geometry column in the first parameter which specifies the geometry column to be searched (the second parameter specifies a query window). The operators can be used only in the WHERE clause or in a subquery and their parameters need not to be in the same coordinate system.

Spatial procedures and functions are provided as subprograms in PL/SQL packages (e.g., spatial functions in SDO_GEOM [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/SDO_GEOM-reference.html]) and do not require or use a spatial index. These subprograms can be used in the WHERE clause or in a subquery and their parameters must have the same coordinate system.

If an operator and a procedure or function perform comparable operations, and if the operator satisfies your requirements, use the operator (e.g., use SDO_RELATE

instead of SDO_GEOM.RELATE, or SDO_WITHIN_DISTANCE instead of SDO_GEOM.WITHIN_DISTANCE).

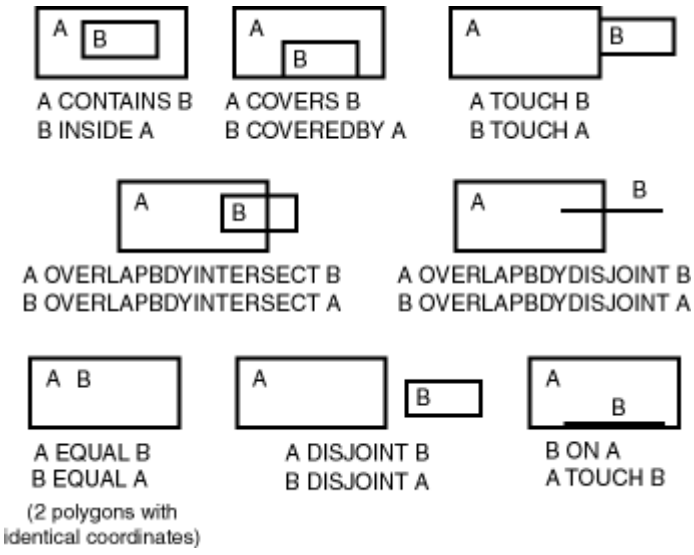
With operators, always specify TRUE in uppercase. That is, specify = 'TRUE', and do not specify <> 'FALSE' or = 'true'.

Spatial Operators

The main spatial operators include:

- `SDO_FILTER(g1,g2)` – Specifies which geometries `g2` may interact with a given geometry `g1`.
- `SDO_RELATE(g1,g2,'MASK=m')` – Determines whether or not two geometries `g1` and `g2` interact in a specified way given by a mask `m`.
- `SDO_WITHIN_DISTANCE(g1,g2,'DISTANCE=d,UNIT=u`) `-- Determines if two geometries` g1 `and` g2 `are within a specified distance` d `(units` u` for geodetic coordinates) from one another.
- `SDO_NN(g1,g2,'SDO_NUM_RES=n,UNIT=u',r)` – Determines the `n` nearest neighbor geometries `g1` to a geometry `g1` with the ability to get their distance (in units `u` for geodetic coordinates) by number `r`.
- `SDO_NN_DISTANCE(r)` – Returns the distance of an object returned by the SDO_NN operator with number `r`.

The mask in the `SDO_RELATE` operator and also in the `SDO_GEOM.RELATE` function is setting a type of topological relationships.



With operators, always specify compare their results to `'TRUE'` in uppercase (that is, specify `= 'TRUE'`, and do not specify `<> 'FALSE'` or `= 'true'`).

It is recommended to use the `/*+ ORDERED */` optimizer hint if the query window (i.e., a geometry in the second parameter) comes from a table.

For more operators, see spatial operators [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-operators-reference.html] in Oracle documentation from where also the following examples are adopted:

```
SELECT c.mkt_id, c.name
  FROM cola_markets c
  WHERE SDO_FILTER(c.shape,
    SDO_GEOMETRY(2003, NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(4,6, 8,8))
    ) = 'TRUE';

SELECT c.mkt_id, c.name
  FROM cola_markets c
  WHERE SDO_RELATE(c.shape,
    SDO_GEOMETRY(2003, NULL, NULL,
      SDO_ELEM_INFO_ARRAY(1,1003,3),
      SDO_ORDINATE_ARRAY(4,6, 8,8)),
    'mask=anyinteract') = 'TRUE';

SELECT c.name FROM cola_markets c WHERE
SDO_WITHIN_DISTANCE(c.shape,
  SDO_GEOMETRY(2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(4,6, 8,8)),
  'distance=10') = 'TRUE';

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

Spatial JOIN Operator

The operator `SDO_JOIN(t1,c1,t2,c2,'mask=m')` performs a spatial join of tables names `t1` and `t2` based on one or more topological relationships of

geometries in columns names `c1` of table `t1` and `c2` of table `t2`. The topological relationships are given by mask `m` with the same mask types as in `SDO_RELATE` operator above.

The example from Oracle spatial operators [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-operators-reference.html] documentation:

```
SELECT /*+ ordered */ a.name, b.name
  FROM TABLE(SDO_JOIN('COLA_MARKETS', 'SHAPE',
                      'COLA_MARKETS', 'SHAPE',
                      'mask=ANYINTERACT')) c,
       cola_markets a,
       cola_markets b
  WHERE c.rowid1 = a.rowid AND c.rowid2 = b.rowid
  ORDER BY a.name;
```

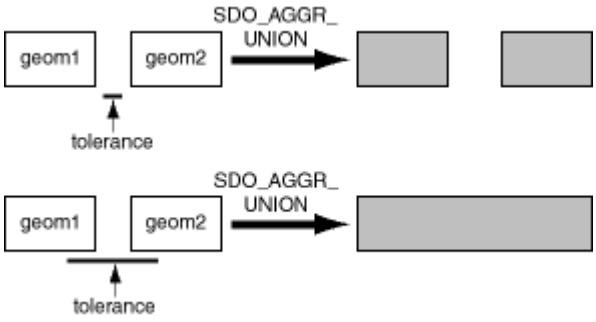
Spatial Functions

The main spatial functions are spatial functions in SDO_GEOM [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/SDO_GEOM-reference.html].

Functions to query spatial relationships:

- `SDO_GEOM.RELATE(g1,'m',g2,t)` – Examines two geometry objects `g1` and `g2` to determine their spatial relationship with a given mask `m` and a particular spatial tolerance [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-concepts.html#GUID-7469388B-6D23-4294-904F-78CA3B7191D3]. If you pass the `DETERMINE` keyword in mask `m`, the function returns the one relationship keyword that best matches the geometries, otherwise, the function returns `'TRUE'` or `'FALSE'`.

The tolerance is a level of precision with spatial data, i.e., the distance that two points can be apart and still be considered the same.



Analytical functions:

- `SDO_GEOM.SDO_AREA(g,t,u)` – Returns the area of a two-dimensional polygon in geometry `g` with tolerance `t` (and units `u` for geodetic coordinates).
- `SDO_GEOM.SDO_LENGTH(g,t,u)` – Returns the length or perimeter of a geometry object `g` . with tolerance `t` (and units `u` for geodetic coordinates).
- `SDO_GEOM.SDO_DISTANCE(g1,g2,t,u)` – Computes the minimum distance between two geometry objects `g1` and `g2` with tolerance `t` (and units `u` for geodetic coordinates), which is the distance between the closest pair of points or segments of the two objects.

Functions to create new geometries:

- `SDO_GEOM.SDO_BUFFER(g,d,t)` – Generates a buffer polygon in distance `d` around or inside a geometry object `g` with tolerance `t` .
- `SDO_GEOM.SDO_CENTROID(g,t)` – Returns a point geometry that is the centroid (a center of gravity) of a polygon, multipolygon, point, or point cluster `g` with tolerance `t` .
- `SDO_AGGR_UNION(SDO_GEOM.SDOAGGRTYPE(g,t))` – An aggregate function which returns a geometry object that is the topological union (OR operation) of the specified geometry objects `g` with tolerance `t` .
- `SDO_AGGR_CENTROID(SDO_GEOM.SDOAGGRTYPE(g,t))` – An aggregate function which returns a geometry object that is the centroid (a center of gravity) of the specified geometry objects `g` with tolerance `t` .

The (distance) buffer of an object consists of all points within the given distance from that object.

For more, see spatial functions in SDO_GEOM [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/SDO_GEOM-reference.html] and spatial aggregate functions [https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-aggregate-functions.html].

Oracle Spatial Data in Java

```
package demo.spatial.examples;

import java.lang.Exception;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Struct;
import oracle.jdbc.pool.OracleDataSource;
import oracle.spatial.geometry.JGeometry;

class Demo2Spatial {

  public static void main (String args[]) {

    try {
      OracleDataSource ods = new OracleDataSource();

ods.setURL("jdbc:oracle:thin:@//gort.fit.vutbr.cz:1521/orclpdb");
      ods.setUser(System.getProperty("login"));
      ods.setPassword(System.getProperty("password"));
      Connection conn = ods.getConnection();

      try {
        JGeometry jgeom = null;
        Statement stmt = conn.createStatement();
        try {

          // reading a geometry from database
```

```
          ResultSet rset = stmt.executeQuery(
            "select shape from cola_markets where mkt_id = 1");
          try {
            if (rset.next()) {
              // convert the Struct into a JGeometry
              Struct obj = (Struct) rset.getObject(1);
              jgeom = JGeometry.loadJS(obj);
            }
          } finally {
            rset.close();
          }
        } finally {
          stmt.close();
        }

        // manipulate the geometry via JGeometry
        System.out.println(jgeom.toStringFull());

        // writing a geometry back to database
        PreparedStatement pstmt = conn.prepareStatement(
          "update cola_markets set shape=? where mkt_id = 1");
        try {
          // convert the JGeometry instance to a Struct
          Struct obj = JGeometry.storeJS(conn, jgeom);
          pstmt.setObject(1, obj);
          pstmt.executeUpdate();
        } finally {
          pstmt.close();
        }
      } finally {
        conn.close();
      }
    } catch (SQLException sqlEx) {
      System.err.println("SQLException: " + sqlEx.getMessage());
    } catch (Exception ex) {
      System.err.println("Exception: " + ex.getMessage());
    }
  }

}
```

To compile and run:

```
javac -classpath .:./lib/ojdbc8.jar:./lib/sdoapi.jar \
  demo/mm/examples/Demo1MmInsert.java
java -classpath .:./lib/ojdbc8.jar:./lib/sdoapi.jar \
  -Dlogin=xnovak99 -Dpassword=*** \
  demo.spatial.examples.Demo2Spatial
```