Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Volume 21, No. 1, ISSN 1726-9679
ISBN 978-3-901509-73-5, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, EU, 2010
Make Harmony Between Technology and Nature, and Your Mind will Fly Free as a Bird
Annals of DAAAM International

IMPORT/ EXPORT OF GIS DATA IN ORACLE SPATIAL

BOICEA, A[lexandru]; BENTU, A[lexandru] S[tefan]; RADULESCU, F[lorin] & STOEAN, N[adia] G[abriela]

Abstract: *This paper presents raster and vector GIS files and the Oracle Spatial database object. The application is made in Java and this paper presents the algorithms for the import, export of raster/vector GIS files and also the conversion of raster to vector and vector to raster data.*
Key words: *Oracle Spatial, database, GIS, raster, vector*

1. INTRODUCTION

This paper address several theoretical and practical aspects of software development aimed at the import and the export of GIS databases in Oracle Spatial. To form the Oracle Spatial database we will use GIS raster files (ESRI, BIL) or GIS vector files (GML, Shapefile). The application connects to the database or GIS files to extract or enter geograic data and for these it uses different algorithms for import and export of geographic data and heuristic algorithms for the conversion between raster and vector data. The user is able to choose exactly how the data will be converted.

Nowadays, on the market there are few applications that assure only the import or export of GIS data into Oracle Spatial database like: SQL Loader, GRASS, Spatial Console or PalentGIS. None of these applications can assure conversion between different data types (raster and vector).

The application can be improved by using multiple GIS files, creating software modules for viewing maps and for editing spatial data. In addition to that, the conversion alghoritms can be optimized in order to make the operations faster.

2. TECHNOLOGIES

Raster files are generally used to store image information, data captured by satellites or other airborne imaging systems. A raster format is reprezented by any type of digital image stored in grids. The raster data is divided into cells, pixels or elements. Cells are organized in arrays and each one has a single value that represents a geographic attribute for the area. The row and column numbers are used to identify the location of each cell within the array (George, 2001). The raster GIS files used for the application are: BIL (binary format) and ESRI (ASCII format)

Vector files are represented by vectorial elements: points, lines, polygons, arcs, string lines etc. Each GIS file type has its own way to represent the vectorial information, in one or more files. The vector GIS files used for the application are: ShapeFile (binary format) and GML (ASCII format).

Oracle Spatial provides an SQL schema and functions that facilitate the storage, retrieval, update and query of spatial data. Oracle Spatial supports the object-relational model for representing geometries. The object that can store geometric data is: MDSYS.SDO_GEOMETRY (Albert, 2007).

3. APPLICATION

This application makes the import and export from an Oracle Spatial database to GIS data files. The application also makes the raster/vector conversion for different GIS file types. To store data in Oracle Spatial, we need two tables: in one we store the metadata (MapInfo) for all the maps and in the others the actual map (Chuck, 2003).

MapInfo		
Name	Variable	Description
MAP_ID	NUMBER(6)	Map primary key
MAP_NAME	VARCHAR2(15)	Map name
MAP_TYPE	NUMBER(1)	Value 0 for a raster map and 1 for a vector one
TABLE_NAME	VARCHAR2(15)	Table name which stores the map. Unique value
XMIN	NUMBER(10,4)	Minimal latitude
XMAX	NUMBER(10,4)	Maximal latitude
YMIN	NUMBER(10,4)	Minimal longitude
YMAX	NUMBER(10,4)	Maximal longitude
ZMIN	NUMBER(10,4)	Minimal altitude
ZMAX	NUMBER(10,4)	Maximal altitude
DESCRIPTION	VARCHAR2(50)	Supplementary information

Tab. 1. Metadata table for the maps

Raster Import:
1. The header from the raster file is read and analyzed
2. Metadata is inserted in *MapInfo* about the new map
3. Is created a new table where the map will be stored (table_name)
4. Every cell are read from the raster cell and converted in *SDO_GEOMETRY* objects
5. Data is inserted in the database

Raster Export:
1. The GIS file is created and the header is filled with metadata
2. Data is read from the table and is converted from *SDO_GEOMETRY* objects into a single value (altitude)
3. Data is inserted in the file at the specific position in the matrix

Vector Import:
1. The map is analyzed and metadata is inserted in *MapInfo*
2. Is created a new table where the map will be stored
3. Every vector is read from the GIS file and is converted in *SDO_GEOMETRY* objects
4. Data is inserted in the database

Vector Export:
1. The GIS file is created
2. Data is read from the table and is converted from *SDO_GEOMETRY* objects into a vector format specific to the vector GIS file type
3. Data is inserted in the database.

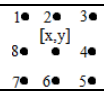
Vector-Raster Conversion:
1. The vector GIS file or map stored the database is analyzed
2. Metadata is inserted in *MapInfo* about the new map
3. Is created a new table where the map will be stored
4. Repeat for every cell in the matrix map
   4.1. Select the vector data that intersects with the cell
   4.2. Calculate the altitude value for the cell: weighted average of the surfaces occupied by the vector data in the cell
   4.3. Convert data into a *SDO_GEOMETRY* object
   4.4. Data is inserted in the database

Vector-Raster Conversion:
1. The map is analyzed and metadata is inserted in*MapInfo*
2. The following algorithm is applied:

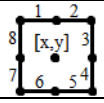
U[x][y]	= 0 - cell is not being used; =1 - cell is being used
A	Matrix that stores the altitude of the map cells
n	Number of raster cells on latitude in the map
m	Number of raster cells on longitude in the map
P	List of cells with the same altitude
N	List of neighbor cells with the same altitude
T	Temporary value, T=(x,y)
F	Polygon resulted from P

Tab. 2. Variables used in the *main* algorithm

main()
1. initialize A, U //U[x][y]={{0}}
2. repeat until sum(U[x][y]==n*m)
  2.1. initialize P,N //P={}, N={}
  2.2. select first x, y where U[x][y]=0
  2.3. U[x][y]=1; P=P union {(x,y)};T=(x,y)
  2.4. repeat
    2.4.1. find all neighbors to T like (nx,ny) where
        A[nx][ny]=A[x][y] and U[nx][ny]=0
    2.4.2. insert neighbors into N
    2.4.3. s=0; P=P union {(nx,ny)} for all (nx,ny)
    2.4.4. if N is not empty
      2.4.4.1. remove first (nx,ny) from N
      2.4.4.2. s = 1;T = (nx,ny)
  2.5. until s=0
  2.6. F=create_poligon(P)
  2.7. convert F to SDO_GEOMETRY
  2.8. insert F into database

P	List of cells that will form the polygon
E	List of all edges created by the cells
V	List of all vertices created by the cells
EU	List of edges necessarily for the polygon creation
nv	Number of neighbor vertices (minimum 3, maximum 8) 
nei	Number of edges that include a point (minimum 2, maximum 4) The edges are numbered by their position 

EI	Edge list that include a point // EI[1..4]
	EI[1]=null if edge in position 1 does not exists
	EI[1]=edge1 if edge in position 1 does exists

nes	Number of edges that surround a point (minimum 3, maximum 8) The edges are numbered by their position 

ES	Edge surround list // ES[1..8]
	ES[1]=null if edge in position 1 does not exist
	ES[1]=edge1 if edge in position 1 does exists

EUO	List of ordered edges necessarily for the polygon creation EUO[edge_nr][1..2] // 1, 2 are the vertices of the edge
FV	List of vertices lists that forms the polygon
AV	Temporarily vertices list // FV[x]=AV
FD	List of directions for FV. FD[x] corresponds to list FV[x] FD[x]=0 for the polygon surface that is included FD[x]=1 for the polygon surface that is excluded

Tab. 3. Variables used in the *create_poligon* function

create_poligon(P)
1. initialize E,V //E={},V={}
2. for each cell in P
  2.1. calculate the edges and vertices of the cell P[x][y]
  2.2. insert in Eand V the edges and vertices that are unique
3. initialize EU //EU={}
4. for every point (x,y) in V
  4.1. calculate nv,nei,nes for (x,y) from V
  4.2. create ES,EI and initialize them //view Table x.x
  4.3. if (nei==2) then
insert into EU at the end, edges where EI[x]!=null, 0<x<5
  4.4. if (nei==3) then
    4.4.1. if (EI[1]=null orEI[3]=null)
insert into EU at the end, edges EI[2], EI[4]
    4.4.2. if (EI[2]=null orEI[4]=null)
    insert into EU at the end, edges EI[1], EI[3]
  4.5. if (nei==4) and ((nv=7) or (np=8 and nei!=8)) then
    4.5.1. if(ES[1]=null or ES[8]=null) then
insert into EU at the end, edges EI[4], EI[1]
    4.5.2. if(ES[2]=null or ES[3]=null) then
insert into EU at the end, edges EI[1], EI[2]
    4.5.3. if(ES[4]=null or ES[5]=null) then
    insert into EU at the end, edges EI[2], EI[3]
    4.5.4. if(ES[6]=null or ES[7]=null) then
    insert into EU at the end, edges EI[3], EI[4]
5. create EUO // EUO={}
6. repeat
  6.1. p0=p1, extract first edge (p1,p2) from EU // p1=(x1,y1)
  6.2. insert into EU at the end, edge (p1,p2)
  6.3. repeat
    6.3.1. extract edge (pa1,pa2) from EU where
    pa1=EUO[last_element][2] or pas2=EUO[last_element][1]
    6.3.2. if (pa1=EUO[last_element][2]) then
    insert into EU at the end, edge (pa1,pa2)
    6.3.3. if (pa2=EUO[last_element][1]) then
    insert into EU at the end, edge (pa2,pa1)
  6.4. until p0=EUO[last_element][2]
7. until MP is empty
8. for every e1=EUO[t] e2=EUO[t+1] //ei=((xi1,yi1),(xi2,yi2))
where t=module(0..size(EUO), size(EUO))+1
  8.1. if (e1[2]=e2[1]) and
      (e1[1][1]=e2[2][1] or e1[1][2]=e2[2][2]) then
    8.1.1. remove from EUO: EUO[t+1], EUO[t]
    8.1.2. insert into EUO on position t, edge (e1[1],e2[2])
9. i=0, create FV, FD // FV={{}}, FD={}
10. repeat
  10.1. i=i+1, create TV // TV={}
  10.2. extract first edge e=(p1,p2) from EUO //EUO[1]=e
  10.3. p0=p1
  10.4. if(p1[1]=p2[1] and p1[2]<p2[2]) or
        (p1[2]=p2[2] and p1[1]>p2[1])
      then FD[i]=0; else FD[i]=1
  10.5. insert into TV vertex p1 // TV[1]=p1
  10.6. repeat
    10.6.1. extract first edge (p1,p2) from EUO
    10.6.2. insert into TV at the end, vertex p1
    10.6.3. if (p0=p2) then
      10.6.3.1. insert into TV at the end, vertex p2
      10.6.3.2. insert into FV at the end, list TV
          //FV[size(FV)+1] = TV
  10.7. until (p0=p2)
11. until EUO is empty
12. return (FV,FD)

4. CONCLUSIONS

This paper tried to emphasize the importance of software that allows conversion between different types of spatial data and assures import and export of GIS data in Oracle Spatial Database. For the application were chosen all the different types of GIS files: both raster and vector type and ASCII and binary formats. The purpose of selecting these files was to cover all types of spatial data conversions that may exist. Nowadays there isn't an application that facilitate all these functionalitys (import, export, convert) and neither one that can make a conversions between raster and vector formats.

5. REFERENCES

Albert, G.; Ravi, K. & Euro, B. (2007). *Pro Oracle Spatial for Oracle Database 11g*, Apress, 978-1-59059-899-3, USA

Chuck, M. (2003). *Oracle Spatial User's Guide and Reference*

George, B. &Korte, P. (2001). *The GIS Book*, Fifth Edition, Ed. OnwordPress, 0-7668-2820-4, Canada

*** (2010) http://en.wikipedia.org/wiki/GIS_file_formats, *Accessed on: 2010-01-10*

***(2001)http://oreilly.com/catalog/orsqlloader/chapter/ch01.html, *Accessed on: 2010-01-12*