Storing GML Documents: A Model-Mapping Based Approach*

Fubao Zhu[1,2], Qianqian Guo[2], and Jinmei Yang[2]

[1] School of Computer Science, Fudan University, Shanghai 200433, China
fbzhu@fudan.edu.cn
[2] School of Computer and Communication Engineering,
Zhengzhou University of Light Industry, Zhengzhou 450002, China

Abstract. Geography Markup Language is a de facto standard developed by OGC to standardize the representation of geographical data in XML, which makes the exchanging and sharing of geographical information easier. With the popularity of GML technology, more and more geographical data is presented in GML format. This causes the problem of how to efficiently store GML data to facilitate its management and retrieval. An approach to store and query GML document based on model-mapping is proposed in this paper. The proposed approach mainly focus on non-schema GML documents, but it is also applicable to GML documents with corresponding schemas. A GML document is first parsed, and a document tree is generated. Then the tree nodes are analyzed and processed, and the schema mapping is established for storing GML documents into object-relational database with structural information preserved. Spatial data analysis and non-spatial data query are supported on database for document. Experiments show that the proposed approach is feasible and efficient.

Keywords: Geography Markup Language; Schema; Model-mapping; Storage; Object-relational Database.

1 Introduction

The Geography Markup Language (GML) is the XML grammar defined by the Open Geospatial Consortium (OGC) for expressing geographical features. It is an international standard for data encoding, transmission, storage and release. It is applied to geographical data sharing, exchanging and integration on Internet. With the development of GML, the growing ability of GML has solved inconsistent formats of spatial data providing the data expression including data structure and semantics. It conforms to the requirements of Web and makes it much easier for data exchange, integration, and sharing between different systems.

* This work was supported by the National Natural Science Foundation of China (No. 60873040), China 863 Program (No. 2009AA01Z135), Key Science and Technology Project of Education Department of Henan Province (No. 2010B520033), and Doctoral Research Fund of Zhengzhou University of Light Industry (No. 2008BSJJ012).

With the extensive application of GML, a growing number of geographic information is described by GML format. Like the XML encoding, GML uses text format to express geographic information. Because text file of GML is quite large, it is impossible to manage geographical data well with good spatial information query, spatial data analysis, access and concurrency control, etc. The effective management of GML data becomes an urgent problem which is aimed to facilitate analysis of spatial operations and promote the sharing of spatial data and GIS interoperability. GML document will be stored in the object-relational database where GML non-spatial attributes act as common fields, spatial attributes act as objects. It is an available solution to manage GML data by object-relational databases.

GML application schema provides the format specification and semantic constraints for the preparation of GML document. The document which has schema can implement storage through mapping mechanism between GML schema and database schema. This method is called structure-mapping [1]. However, in many applications, the schema of GML document is not clear, or even has no schema. The major problem here is how to store GML document in the database and meet different query requirements. Such method is called model-mapping [3,4,5,6]. It does not use any schema of stored documents. Instead, it parses the original document into a document tree in memory, creating a generic (object) relational schema to store the document's structure and data. Therefore we propose a GML document storage method based on model mapping. GML document is regarded as a tree composed of the element nodes, attribute nodes and geometry nodes. The process of tree nodes and edges help to store GML document into the database and to support spatial information analysis and non-spatial information inquiries.

2 Related Work

At present, a lot of work focused on the XML documents based on model-mapping and the storage management of database [3,4,5,6]. The work parsed the XML document into a tree, and processed differently nodes and edges of document tree.

The method of XParent [6] sumed up the three methods of Edge [3], Monet [4] and XRel [5]. It divided those into the Edge-Oriented [3,4] method and the Node-Oriented [5] method. And it also used four tables to represent the data and structure information in XML document, which included path table, data path table, element table and data table. It made full use of advantages of Edge, Monet and XRel, and provided a good query capability.

Córcoles [7] analyzed the performance of storing GML documents of LegoDB [2], Monet and XParent. And it proved that LegoDB had the optimal performance in storage and query, as well as LegoDB was better to support the spatial information processing. But LegoDB was based on the structure mapping and it was not suitable for non-schema GML document storage.

All the above methods [2,3,4,5,6] are designed for XML document storage, and without considering the spatial data information of GML document, thus they were not suitable for GML document storage. At present, there is no literature related to storage management technologies and methods about the non-schema GML document. Since GML is based on XML, methods of XML storage management can help a lot on the study of non-schema GML document storage management. This paper makes full use of these advantages of non-spatial information storage, and expands its storage capability of spatial information. It has proposed the model-mapping method which is based on nodes and edges to store non-schema GML documents.

3 Model-Mapping Storage Method Based on Nodes and Edges

This section details the architecture of GML document storage and GML document data model. According to an illustration of GML document, we describe the schema mapping method from nodes and edge of GML document tree to database. The storage time-consuming of different document size and different data type is presented to validate the proposed storing method.

3.1 GML Document Storing Architecture

The types of data in the GML document are point, line, polygon and so on. Separating different data effectively and storing data into the database by the processor is the main issue of the paper. The architecture is shown in Fig. 1.

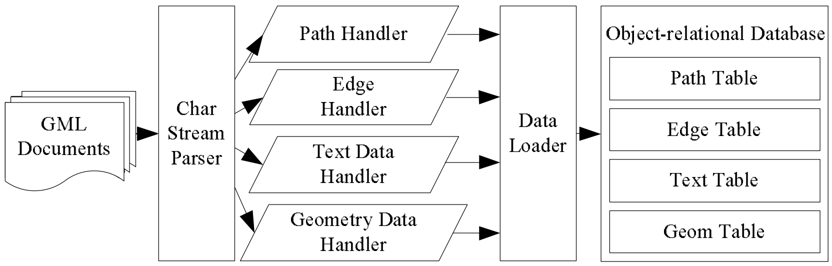

Fig. 1. Architecture of GML Storage

The path data, edge data, text data and geometry data of the document are parsed by the character stream parser from the GML document. The data is processed by the corresponding processor, and then stored in the corresponding table in object-relational database by the data loader, to realize analysis and computation of spatial data and query and manipulation of non-spatial data.

3.2 GML Document Data Schema

A complete GML document can be viewed as a tree composed of a number of nodes. Figure 2 shows the structure of document tree of document instance described in Prev verse, including root node, element nodes, attribute nodes, text nodes and geometry nodes. Root node is a virtual node which points to root element node of GML document. Element node represents element in the GML document which is named after the label of element and contains more than one text node and child element node. Text node only contains string information which represents non-spatial data in GML document and does not contain any child nodes. Elements in GML document can contain a number of attributes which are shown as attribute nodes in the document tree. Attribute nodes are composed by attribute name and attribute values which have no child nodes. Geometry is a special element node which is used to represent the geometric information of spatial data in GML document. In Fig. 2, symbols $\triangle$, $\bigcirc$, $\square$, $\diamondsuit$ separately represent the element node, attribute node, text node and geometry node. Note that in GML document tree, we do not draw a specific spatial information which is represented in geometry node. When the parser encounters a geometry node, it will deal with it as an object.

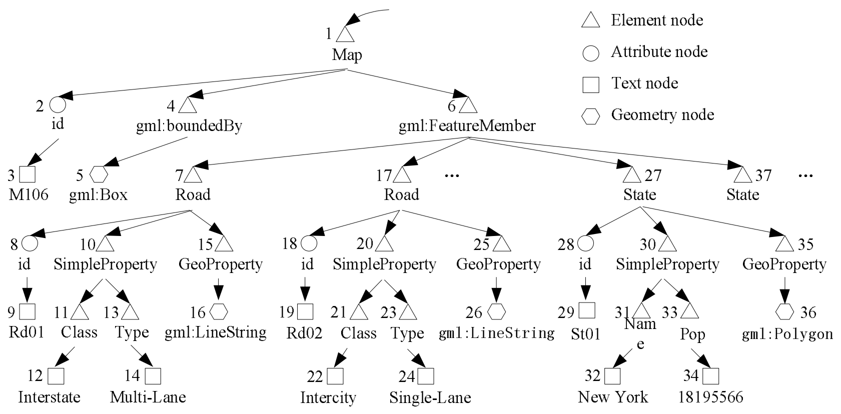

Fig. 2. GML document tree

3.3 Constructing of GML Document Tree

GML document is stored as a text file. We use the follow steps to store the data and structural information of the document into the database. The first step, it should parse the GML document. The second step, use the result of parsing to build the document tree. Next, use IDs to number nodes of different types of the tree. Last, determine the order of nodes which have the same parent and the same label in siblings. The algorithm of document tree construction is shown in Algorithm 1.

Algorithm 1. Constructing GML document

```
Result: CreateGMLDocTree(gb)
input  : GMLDocument gd
output: GMLDocumentTree gdt
parse GML document gd as DOM tree dt;
get the root node root of DOM tree dt; let nID and ord be id and ordinal of
nodes in dt, respectively
GenerateDocTree (root); // Generate document tree
Procedure GenerateDocTree(node)
if the node type of node is not Geometry then
    assign node id nID to node;
    //Given node to node numbering
    nID=nID+1;
    //Calculated number for the next node
    let pn be parent node of node;
    //Get the parent node
    let ord be ordinal of sibling node which has same name;
    if pn is not null then
        //If the parent node is not null
        calculate the ordinal ord of node in its same-tag-sibling
        assign ord to node; //computer node's ordianal value
    else
        let cns be the child node list of node;//Get the child node
        for each cnsi in cns do GenerateDocTree(cnsi);
        //Recursively process each child node
    end
else
    construct Geometry g of node;
    //Construct geometry object
    calculate nID and ord for g;
    //Calculate the number and order for geometry
end
```

3.4 GML Document Database Model

When GML document has been stored in the database, it should support querying and processing of non-spatial information, and analyzing and calculating of spatial data. To realize the object, an approach of node and edge based Model-mapping is proposed in the paper (Node and Edge based Model-mapping, NEM). NEM defines the label path table, edge table, text table and geometric table in processing nodes and edges of the document tree. The structure of the four tables is shown as follows:

```
LabelPathpathID, pathExpr
EdgedocID, pID, cID
TxtDatadocID, pathID, ordinal, nID, type, value
GeoDatadocID, pathID, ordinal, nID, type, shape
```

Algorithm 2. Storing GML document

```
Result: StoringGMLDocuments($gdt$)
input  : GMLDocumentTree $gdt$
output:  four tables
let $docID$ be the maximum document id of current database;
//Get root node
if $docID$ is not null then
    //Calculated the number of docID
    $docID = docID + 1$;
else
    $docID$ =1;
end
let $root$ be root node of $gdt$;
//Get the root
let $pathID$ be the maximum path id of current database;
//Get the pathID
if $pathID$ is not null then
    //Calculate the number of pathID
    $pathID = pathID + 1$;
else
    $pathID$ =1;
end
TraversalNodes($root$);
Process TraversalNodes(Node $n$)
if type of node $n$ is Element then
    Let $pathExpr$ be label path from root to current node;
    if $pathExpr$ does not exist in table LabelPath then
        $pathID = pathID + 1$; insert $pathID$ and $pathExpr$ into table
        LabelPath;
    else
        endif
    end
else
    endif
end
let $pn$ be parent node of $n$;
//Get the parent node
if $pn$ is not null then
    ///if the parent node is not null
    let $pID$ and $cID$ be node id of $pn$ and $n$, respectively;
    insert $docIDC$, $pID$ and $cID$ into table Edge;
else
    endif
end
if  type of $n$ is Text or Attribute then
    calculate pathID and get ordinal, type and value of node $n$;
    insert the above values into table TxtData;
else

end
if type of $n$ is Geometry then
    calculate pathID and get ordinal, type and value of node $n$; insert the above
    values into table into table GeoData;
else
    endif
end
let $cns$ be the child node list of $n$;
for each $ni$ in $cns$ do TraversalNodes ($ni$);
return LabelPath, Edge, TxtData and GeoData;
```

Table 1. LabelPath table

pathID	pathExpr
1	/Map
2	/Map/id
3	/Map/boundedBy
4	/Map/boundedBy/gml:Box
5	/Map/FeatureMember
6	/Map/FeatureMember/Road
7	/Map/FeatureMember/Road/id
8	/Map/FeatureMember/Road/SimpleProperty
9	/Map/FeatureMember/Road/SimpleProperty/Class
10	/Map/FeatureMember/Road/SimpleProperty/Type
11	/Map/FeatureMember/Road/GeoProperty
12	/Map/FeatureMember/Road/GeoProperty/gml:LineString
13	/Map/FeatureMember/State
14	/Map/FeatureMember/State/id
15	/Map/FeatureMember/State/SimpleProperty
16	/Map/FeatureMember/State/SimpleProperty/Name
17	/Map/FeatureMember/State/SimpleProperty/Pop
18	/Map/FeatureMember/State/GeoProperty
19	/Map/FeatureMember/State/GeoProperty/gml:Polygon

Table 2. Edge table

docID	pID	cID	docID	pID	cID
1	1	2	1	17	18
1	1	4	1	17	20
1	1	6	1	17	25
1	2	3	1	18	19
1	4	5	1	20	21
1	6	7	1	20	23
1	6	17	1	23	24
1	6	27	1	25	26
1	7	8	1	27	28
1	7	10	1	27	30
1	7	15	1	27	35
1	8	9	1	28	29
1	10	11	1	30	31
1	10	13	1	30	33
1	11	12	1	31	32
1	13	14	1	33	34
1	15	16	1	35	36

LabelPath table records the label path information of the document tree node, the pathID is the LablePath table's ID and the pathExpr is the corresponding path's expression. Edge table records the document tree edge, docID, pID and

Table 3. TxtData table

docID	pathID	ordinal	nID	type	value
1	2	1	3	T	M16
1	7	1	9	A	Rd01
1	9	1	12	T	Interstate
1	10	1	14	T	Multi-Lane
1	7	2	19	A	Rd02
1	9	2	22	T	Intercity
1	10	2	24	T	Single-Lane
1	14	1	29	A	St01
1	16	1	32	T	Clark Fork
1	17	1	34	T	Columbia

Table 4. GeoData Table

docID	pathID	ordinal	nID	type	shape
1	4	1	5	Polygon	GEOM (Box)
1	12	1	16	Line	GEOM (LineString)
1	12	2	26	Line	GEOM (LineString)
1	19	1	36	Polygon	GEOM (Polygon)

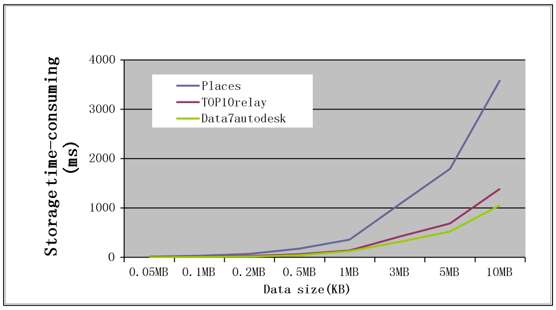

Fig. 3. Model-independent GML data storage time-consuming

cID which separately represent IDs of document identification, parent node and child node in the document. TxtData table records the information of text node and attribute node, ordinal represents the order of child nodes which have the same label path and parent node, nID represents the node's number, type represents the node's type (text node or attribute node), value represents the node's value, GeoData table records the information of the geometry node, type represents the type of the geometry node, shape represents the information of the geometry node. The algorithm of the data in the document tree stored in the database is shown in Algorithm 2. When the document tree in Fig. 2 stored in the database, you can use the node, edge and path information in Table 1 to Table 4 to describe.

3.5 The Experimental Analyzing of GML Document Data Storing Time

We have done the experiments on 24 different GML documents with its size range from 50KB to 10MB, and figured out the time-consuming of storage. The GML documents contains 8 places documents of Point Layer, 8 TOP10relay documents of Line Layer and 8 Data7autodesk documents of Plane Layer. It can be seen from experiments' results, shown in Fig. 3, that the proposed storage method has an obvious advantage on the time-consuming of storage.

4 GML Query Processing

NEM supports non-spatial data query and spatial data analysis and computation. We use GML-QL [8] to write query language. But GML-QL is just a query language used on GML document, and cannot be directly executed on the database, so it must be converted to the corresponding SQL query language. In the paper, Oracle Spatial is used to store GML documents. In order to determine the relationship between grandparent and grandchild nodes quickly, we use a self-defining function UDF_PARENT. Querying-transforming is a complex process [9], and it is closely related to the data storage schema in database [10,11]. The paper defines four tables, separately storing node paths, parent-child relationships between nodes, the paths and the values of text nodes and geometry nodes. The four tables are suitable for the conversion of the XQuery based on path expression and SQL query extended to the database. Through the multi-geometry spatial operation and analysis, GML query queries the non-spatial information which satisfy spatial relations. GML query is shown in the following part.

```
FOR $r IN document("Map.xml")//Road,
    $s IN document("Map.xml")//State
WHERE Cross($r/GeoProperty/gml:LineString,
         $s/GeoProperty/gml:Polygon) == 1
RETURN
<RiverStates>
 <Rid>$r/@gml:id</Rid>
 <sname>$s/SimpleProperty/Name</sname>
</RiverStates>
```

The corresponding conversed SQL query is shown as following:

```
SELECT t1.value, t2.value
FROM TxtData t1, TxtData t2, GeoData t3, GeoData t4,
     Path p1, Path p2, Path p3, p4, Edge e1, Edge e2
WHERE p1.pathExpr LIKE '%/Road/gml:id'
AND p2.pathExpr LIKE '%/State/SimpleProperty/Name'
AND p3.pathExpr LIKE '%/Road/GeoProperty/gml:LineString'
AND p4.pathExpr LIKE '%/State/GeoProperty/gml:Polygon'
AND e1.cID = t1.nID AND e2.cID = t2.nID
AND t1.type = 'A' AND t1.pathID = p1.pathID
AND t2.pathID = p2.pathID
AND t3.type = 'Line' AND t3.pathID = p3.pathID
AND t4.type = 'Polygon' AND t4.pathID = p4.pathID
AND SDO_RELATE(t3.shape, t4.shape,
'mask=ANYINTERACT querytype=WINDOW') = 'TRUE';
```

5 Experimental Analysis

GML document contains both non-spatial data and a large number of spatial data. Parsing the spatial data and storing the data as spatial data type in the database are key issues of storing GML document [12]. So we use Oracle xml-parserv2 and sdoutl in Oracle Spatial to realize the storing method proposed in the paper. The architecture of GML storage is shown in Fig. 4.

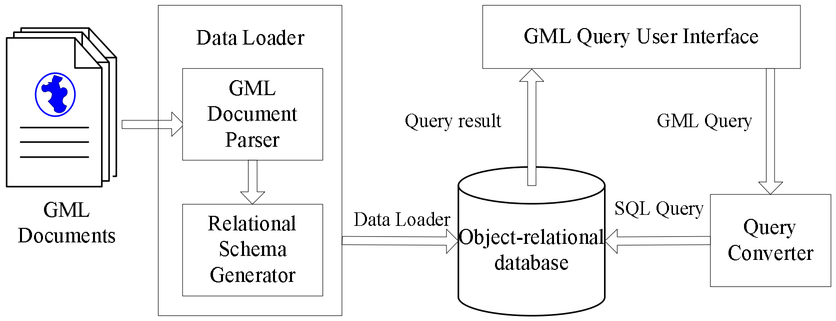

Fig. 4. The architecture of model-driven GML document storage

Data Loader consists of GML document parser and relational schema generator. Parser parses the input GML documents, and constructs document tree. At the same time, DOMParser and JGeometry components are used, DOMParser parses the document as a DOM tree, and JGeometry constructs the geometry nodes in DOM tree as a spatial data object which can be stored directly in database. Relational schema generator traverses the document tree, and separately writes the tree nodes, node direct edges, node path information into the corresponding data table. Query converter converts the input GML query into the SQL query which can run directly in database.

In the experiment, we compare the information in database and documents which contains different sizes and different spatial types. Through the analysis of the data from the instance in ArcView 3.2 and MapInfo 6.0, the analysis of querying time is shown in Fig. 5 and Fig. 6.

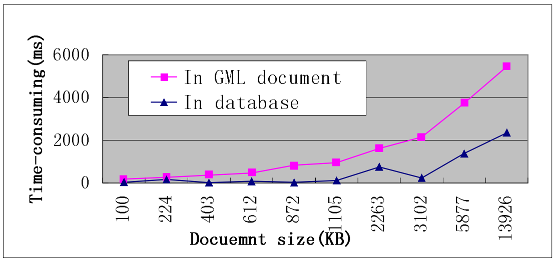

Fig. 5. The same query in GML document and database

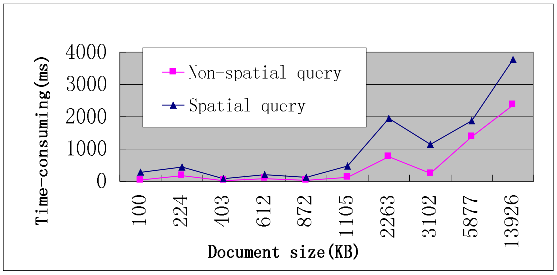

Fig. 6. Spatial query and non-spatial query in database

6 Conclusion

More and more GML documents are created with the wide application of GML technology, and it is important to manage these GML documents. An approach based on node and edge is proposed in the paper in order to store non-schema GML document into object-relational database and model-mapping storage. Firstly document parser converts the GML document into the corresponding document tree, and then constructs schema mapping from GML document to the object-relational database through analyzing and processing the node path, node type, and the parent-child relationship between nodes. The structural information of the document is stored as well. Spatial data analysis and non-spatial data query are the supported through join operations between tables.

References

1. Cox, S., Daisey, P., Lake, R., Portele, C., Whiteside, A.: Geography Markup Language (GML) Implementation Specification, OpenGIS Consortium, version 3.0 (2003)
2. Bohannon, P., Freire, J., Roy, P., Siméon, J.: From XML schema to relations: A cost-based approach to XML storage. In: Proceedings of ICDE, San Jose, California, USA, pp. 64–75 (2002)

3. Florescu, D., Kossmann, D.: A performance evaluation of alternative mapping schemes for storing xml data in a relational database. Technical Report, No. 3680, INRIA, France (1999)
4. Schmidt, A., Kersten, M.L., Windhouwer, M., Waas, F.: Efficient relational storage and retrieval of XML documents. In: Suciu, D., Vossen, G. (eds.) WebDB 2000. LNCS, vol. 1997, pp. 137–150. Springer, Heidelberg (2001)
5. Yoshikawa, M., Amagasa, T.: XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology 1(1), 110–141 (2001)
6. Jiang, H., Lu, H., Wang, W., Yu, J.X.: Path materialization revisited: An efficient storage model for XML data. In: Proc. of ADC, Melbourne, Victoria, Australia, pp. 85–94 (2002)
7. Córcoles, J.E., González, P.: Analysis of different approaches for storing GML documents. In: Proc. of ACM GIS, McLean, Virginia, USA, pp. 11–16 (2002)
8. Vatsavai, R.R.: GML-QL: A spatial query language specification for GML (2002), `http://www.cobblestoneconcepts.com/ucgis2summer2002/vatsavai/vatsavai.htm`
9. Krishnamurthy, R., Kaushik, R., Naughton, J.: XML-to-SQL query translation literature: The state of the art and open problems. In: Proc. of XML Database Symposium, Berlin, Germany, pp. 1–18 (2003)
10. Sun, H., Zhang, S., Zhou, J.: XQuery-to-SQL translating algorithm with little dependence on schema mapping between XML and RDB. In: Proc. of CSCWD 2004, Xiamen, China, vol. 1, pp. 526–531 (2004)
11. Grinev, M., Pleshachkov, M.: Rewriting-based optimization for XQuery transformational queries. In: Proc. of IDEAS 2005, pp. 163–174 (2005)
12. Long, W.X., Hu, C.: An Effective Storage Mode for GML Spatial. In: Proc. of Urban Geotechnical Investigation Surveying, pp. 31–35 (2009)