基于Oracle的空间拓扑管理研究*)

刘峻明　朱德海　张晓东
(中国农业大学信息与电气工程学院　北京100094)

摘　要　空间拓扑关系是GIS的一个重要内容。本文提出了基于Oracle的空间拓扑管理模型。首先，将拓扑元素定义成对象，并列出了这些对象所应该具有的最基础的属性。其次，定义一个拓扑元数据表来对拓扑相关的内容进行描述，拓扑元数据表将起到一个管理和信息导航的作用。最后，讨论了基于该模型如何建立拓扑元素索引的问题。
关键词　Oracle，GIS，空间拓扑，RDBMS

The Research of Spatial Topologically Structured Management Based on Oracle

LIU Jun-Ming　ZHU De-Hai　ZHANG Xiao-Dong
(College of Information and Electric Engineering，China Agricultual University，Beijing 100094)

Abstract　Spatial topology is an important concept. In this paper, a spatial topologically structured management model is described and implemented within Oracle. Firstly, topology features are modeled as objects, which have properties and methods. Secondly, a metadata table is defined to store information related to topology and it also plays a role of topology manager and navigator. Finally, we describe how to create the index for topology features.
Keywords　Oracle, GIS, Spatial Topology, RDBMS

1. 引言

自上世纪90年代以来，随着计算机软硬件的迅速发展和Internet的广泛应用，GIS从一个专业的图形处理平台逐渐转变成基础数据处理平台的一个重要部分。将空间数据与它的属性数据进行集成化管理也逐渐被GIS业界所认识，成为GIS领域中很重要的一个研究课题。

在传统的GIS系统中，对空间数据与属性数据分别进行存贮与管理，两者通过一个关键码来建立关联。空间数据的存贮是基于文件的，不同的GIS厂商的格式互不兼容，没有统一的标准；属性数据的存贮则通常借鉴或者直接采用基于文件的简单关系型数据库系统(如DBASE)来完成，当需要与存贮在大型数据库中的其他相关数据建立关联时，通常采用ODBC、JDBC、ADO等通用的数据库接口。所以在传统的GIS系统中，空间数据与属性数据的关系是不密切的。这种传统方式具有许多优点(如基于文件的存贮格式可以被高效地访问等)，同时也存在许多问题，而且这些问题随着IT业的发展愈加变得突出。这些问题主要有：

1)传统GIS系统在一定程度上割裂了空间数据与属性数据的关系，必须在应用程序一级来保证两者之间的一致性和数据完整性。

2)分离的管理方式使用户难以通过一致的语法来访问空间数据与属性数据。而如果由大型RDBMS来管理，可以使用开放的SQL来访问空间数据。

3)基于文件的方式使得空间数据安全性上无法得到保证，不能设定用户访问级别，也没有一个非常有效的备份机制。而大型的关系型DBMS恰恰在这方面做得非常成功，可以设定多种多级别的权限，可以进行异地备份、灾难恢复等。

4)无法充分满足数据共享的需要。各自不同的GIS系统有着各自不同的空间数据专有结构，使得在实现数据共享和互操作上存在天然缺陷。而采用开放的支持SQL的大型RDBMS平台则避免了专有数据结构，从而可以更好地实现数据共享和服务集成。

所以，近年来不断地有GIS厂商和数据库厂商投入力量来研究空间数据与属性数据进行集成管理的问题。并且已经取得了一些成果。目前几个大型的RDBMS数据库如Oracle、Informix、DB2，都已经实现了对几何数据的存贮。它们定义了点、线、多边形等基础几何特征数据类型，并定义了针对这些类型的一些操作功能。

然而，尽管很多产品都拥有了基础的空间数据存贮和管理能力，但主要针对的是无拓扑的空间实体，在对空间数据的拓扑关系处理上都缺乏必要的能力，拓扑关系应该如何存贮和管理的问题都没有体现出来。大家知道，拓扑关系是GIS中非常重要的一个概念，很多空间分析功能必须依赖于拓扑关系才能做到，比如路径搜索，而有些空间分析则在有拓扑关系时将极大地提高效率。建立拓扑关系也有利于维护空间数据的一致性和完整性，尤其在处理多边形的公共边问题上。另外，保存拓扑信息将大量减少空间矢量数据的存贮量，因为它不需要对多个对象共有的数据进行重复存贮。在农业和土地信息管理方面，拓扑关系更是一个不可回避的问题。

有关如何在RDBMS中实现空间数据存贮和索引的论文有很多，但有关专门讨论如何在RDBMS中实现空间拓扑管理的正式发表的论文，本文作者并未发现。在产品方面，英国的Laser-Scan公司开发出了基于Oracle 9i Spatial的空间拓扑处理技术Radius Topology，它利用Oracle Server对Java的充分支持能力和数据库系统的操作触发机制来实现空间数据的拓扑关系处功能[3]。但是有关Radius Topology技术的细节未见公开发表。

*)本论文受国家十五科技攻关项目“农业信息化技术研究”(2001BA513B3)资助。**刘峻明**　博士，讲师，主要从事地理信息系统、农业信息系统方向的研究。**朱德海**　博士，教授，博士导师，主要研究方向：地理/国土资源信息系统，农业资源与环境信息系统。**张晓东**　博士，副教授，硕士导师，主要从事GIS技术及其应用开发、决策支持系统研究等方面的研究。

本文正是基于这种状况，提出了在 Oracle 支持下的拓扑管理的基本模型。

2. 基础拓扑模型

作为一个初步的探索，我们主要考虑了基本的空间拓扑关系实现的问题。

空间拓扑关系描述了几何物体在空间下的连续和相邻特征[4]。对空间对象进行相邻、相连、包含等空间运算是非常复杂的。拓扑的数据结构可以使这些 CPU 时钟花费很大的几何运算简化为简单的联接运算、代数运算。一幅图的拓扑可以用一维或二维的拓扑元素来表达，分别用于表示网络和面拓扑。

网络拓扑包含两个基本拓扑元素类型：结点（NODE）和边（EDGE）；面拓扑则还包含面（FACE）。

结点（NODE） --- 0维拓扑元素

边（EDGE） --- 1维拓扑元素

面（FACE） --- 2维拓扑元素

在网络拓扑中，线状特征由一个或多个 EDGE 来构成。NODE 则形成于两个线状地物交叉处或者根据拓扑规则形成了某一个点地物与线地物的重合处。一个 EDGE 的起始点和终止点也看成一个 NODE。

面拓扑中的 FACE 由形成封闭环的 EDGE 来构建，可以看成是一幅平面图被它所有的 EDGE 划分之后的一个小斑块，这些小斑块不能相互重叠。

一幅地图最初只是由没有任何拓扑关系信息的线组成的，在建立拓扑关系之后才能形成 EDGE、NODE 和 FACE。

3. 关系数据库平台基础

前面已经提到，有许多大型的关系型数据库都已经能够对简单的空间实体对象进行存贮和管理。由于 Oracle 是著名的大型关系型数据库系统，占有巨大的市场份额，许多数据服务系统都是以它作为平台，所以在基于 Oracle 来建构空间拓扑模型具有很强的代表性和实用性。从技术角度来看，Oracle 也具有很好的技术基础。

Oracle 具有非常强的面向对象特征。近几年来，Oracle 在原来的纯关系型 DBMS 基础上融入了大量的面向对象的特性，结合了 RDBMS 和 OODBMS 的优点，成为一种新型的 DBMS，称为对象关系数据库管理系统（ORDMBS）。用户可以自定义数据类型，针对这些类型定义操作，也可以基于类型和函数建立索引。

自 Oracle 8i 开始还增加了一个 Spatial 模块[2]，它是用来解决空间数据问题的专门模块，它利用了面向对象的特性，使 Oracle 能够处理用空间对象数据类型表示的空间信息，也能够处理使用空间索引方法和函数存取或操作的空间信息。Oracle 提供了一个特殊的方案 MDSYS 来用于存贮和管理空间信息相关的数据库对象，如数据类型、元数据表、函数、包、视图等等。

Spatial 中最重要的一个数据类型是 SDO_GEOMETRY，用于表达几何特征类型，它的定义如下：

```
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);
```

Oracle Spatial 也定义了 SDO_GEOMETRY 中用到的 SDO_POINT_TYPE、SDO_ELEM_INFO_ARRAY 和 SDO_ORDINATE_ARRAY类型：

```
CREATE TYPE sdo_point_type AS OBJECT (
    X NUMBER,
    Y NUMBER,
    Z NUMBER);
CREATE TYPE sdo_elem_info_array AS VARRAY (1048576) of
        NUMBER;
CREATE TYPE sdo_ordinate_array AS VARRAY (1048576) of
        NUMBER;
```

SDO_GEOMETRY 类型中各项的意义是：

SDO_GTYPE 是几何特征的类型。它所支持的类型与 OGIS 的 Simple Features for SQL 规范中定义的的几何对象模型一致。

SDO_SRID 用于表示几何特征所采用的空间参照系统。

SDO_POINT 用于单点几何特征的存贮，主要起到一个优化存贮的作用。如果使用 SDO_POINT，那么 SDO_ELEM_INFO 和 SDO_ORDINATES 必须为 NULL。

SDO_ELEM_INFO 用于描述构成几何特征的各个子部分的特征和信息。

SDO_ORDINATES 用于存贮几何特征的坐标信息。

Spaital 使用专门的一个元数据表 SDO_GEOM_METADATA_TABLE 来描述空间图层的最基本的元数据信息，存放在 MDSYS 方案下。为了屏蔽信息和向后兼容，它还定义了基于该元数据表的一个视图 USER_SDO_GEOM_METADATA。在大多数情况下，我们只需要访问这个视图而不必访问基表。该视图中包含了维数、空间范围、精度等内容。

它的定义如下：

```
CREATE VIEW uer_sdo_geom._metadata IS
(
table_name VARCHAR2(32),
column_name VARCHAR2(32),
diminfo MDSYS.SDO_DIM_ARRAY,
srid NUMBER
);
```

table_name 表示空间图层表的名字；column_name 表示存贮几何特征的列的名字；diminfo 保存了图层的维数、空间范围和精度等信息；srid 则保存图层的空间参照系统的 ID。

每当用户创建一个空间数据表，就必须插入一条元数据信息记录到 USER_SDO_GEOM_METADATA 中。例如：

```
INSERT INTO user_sdo_geom_metadata VALUES (
'landblock', 'geom', MDSYS.SDO_DIM_ARRAY( MDSYS.SDO_
DIM_ELEMENT ('x', 110, 120, 0.1), MDSYS.SDO_DIM_
ELEMENT('y', 30, 40, 0.1)), 8307);
```

8307表示经纬度，采用 WGS 84系统，它在 CS_SYS 表中有定义。

4. 基于 Oracle 的拓扑管理模型

我们按照以下思路建立基于 Oracle 的空间拓扑管理模型。

首先，我们将拓扑元素定义成对象。我们知道，每个对象都具有属性和方法。在下面的具体模型中，我们列出了这些对象所应该具有的最基础的属性。其中拓扑元素对象的几何特征信息采用 SDO_GEOMETRY 对象来表达。

其次，我们将定义一个拓扑元数据表来对拓扑相关的内容进行描述。拓扑元数据表将起到一个管理和信息导航的作用，它与 Spatial 的元数据表 SDO_GEOM_METADATA_TABLE 表结合起来描述空间拓扑信息。

最后，针对这个模型，我们讨论了一下如何建立拓扑元素索引的问题。

下面我们详细描述了我们模型的具体内容。

4.1 拓扑元素对象的定义

根据基本的空间拓扑结构，我们定义出 EDGE、NODE、

FACE 等几个拓扑元素对象。为了查找和处理的方便快速，我们使用一个正整数来作为每个拓扑元素的 ID。ID 作为空间数据表的单独字段存在，不和其他拓扑信息捆在一起，所以下面的拓扑对象类型都没有包含 ID 项。下面给出拓扑元素对象定义的基本模型。

首先定义两个数组类型 edge_id_array 和 part_start_index_array，edge_id_array 用于存贮从一个 NODE 引出来的 EDGE 的 ID 集合或者是围成一个 FACE 的 EDGE 的 ID 的集合，part_start_index_array 用于存贮围成 FACE 的各个环的起始 EDGE 在 edge_id_array 中的位置。定义如下：

```
CREATE TYPE edge_id_array AS VARRAY (10000) of
        NUMBER;
CREATE TYPE part_start_index_array AS VARRAY (1000) of
        NUMBER;
```

一个 EDGE 包含 ID、左右 FACE、起始和终止、EDGE 几何特征等几项内容。它的定义如下：

```
CREATE TYPE topo_edge AS OBJECT
(
left_face_id NUMBER,
right_face_id NUMBER,
from_node_id NUMBER,
to_node_id NUMBER,
geometry SDO_GEOMETRY
);
```

left_face_id 和 right_face_id 分别是 EDGE 的左右多边形的 ID；from_node_id 和 to_node_id 则分别表示 EDGE 起始和终止结点的 ID；geometry 保存 EDGE 所对应的几何特征，为了处理上的方便，也可以将 geometry 独立出来作为单个字段存在。

一个结点包含 ID、结点位置、交汇到该结点的 EDGE 等信息。它的定义如下：

```
CREATE TYPE topo_node AS OBJECT
(
position SDO_GEOMETRY,
edges EDGE_ID_ARRAY
);
```

position 表示结点的位置，实际上，根据拓扑关系，position 信息可以由 edge 得来，但因为 position 比较简单，放到对象结构中可以加快拓扑查询性能。edges 表示交汇到该结点的所有 EDGE 的 ID 集合，当 NODE 是 EDGE 的起始点时，ID 取正数，否则取负数。对于将单独的一个点状特征也看成 NODE 情况，那么只有这几项信息是不够的，需要扩充，这里不再赘述。

一个面是由多个环(1个外环和若干个内环)来围成的，而环是由若干个不交叉的 EDGE 首尾相连围成的。所以面的拓扑信息主要包含围成面的 EDGE 的构成信息。环是有方向的，Oracle Spatial 在定义 SDO_GEOMETRY 时约定，外环采用逆时针方向，内环采用顺时针方向，我们在这里继承了这一约定。

```
CREATE TYPE topo_face AS OBJECT
(
parts PART_START_INDEX_ARRAY,
edges EDGE_ID_ARRAY,
)
```

parts 用于存贮各个环的起始 EDGE 在 edgs 中的位置；edges 则存贮围成 FACE 的所有 EDGE 的 ID，当 EDGE 的走向与环的方向一致时，ID 取正数，否则取负数。

4.2 用元数据表管理拓扑图层

通常，DBMS 中会存在许多的元数据表，用于保存数据库中的所有数据的描述信息，如表、属性、类型等等。同样，要支持拓扑关系，那么拓扑有关的一些结构信息就必须保存起来，供应用程序访问和使用。例如：拓扑层名、边界表名、拓扑多边形表名、节点表明以及相关的一些属性，如单位、比例尺等等。我们将这些信息保存在一个元数据表中，它可以被 DBMS 服务器端、中间件或者应用程序前端所访问和使用，起到为用户导航的作用。从元数据表可以知道拓扑层各个表之间的相互关系，以及什么信息保存在什么位置。下面是一张保存拓扑元数据信息的元数据表：

```
CREATE TABLE topo_metadata
(
name VARCHAR2(32),
description VARCHAR2(256),
edge_table_name VARCHAR2(32),
edge_topo_column VARCHAR2(32),
edge_id_column VARCHAR2(32),
node_table_name VARCHAR2(32),
node_topo_column VARCHAR2(32),
node_id_column VARCHAR2(32),
face_table_name VARCHAR2(32),
face_topo_column VARCHAR2(32),
face_id_column VARCHAR2(32),
srid NUMBER,
diminfo MDSYS.SDO_DIM_ARRAY
);
```

name 是拓扑图层的名字；description 是拓扑图层的一些描述信息；edge_table_name、node_table_name、face_table_name 分别是 EDGE、NODE、FACE 对应的表的名字；

edge_topo_column、node_topo_column、face_topo_column 分别是 EDGE、NODE、FACE 表中的保存拓扑信息的字段的名字，字段类型分别对应前面定义的各拓扑元素对象类型；

edge_id_column、node_id_column、face_id_column 分别是 EDGE、NODE、FACE 表中的保存拓扑元素 ID 的字段的名字，字段类型是整型；

srid 保存图层的空间参照系统的 ID；

diminfo 保存了整个拓扑图层的维数、空间范围和精度等信息。

通过这个表，可以知道一个拓扑层是由那几个表所共同构成的，通常这个表必须与 mdsys.uer_sdo_geom._metadata 联合起来。

从前面的拓扑信息对象类型定义中可以看出，EDGE 可以直接获得它的几何特征信息；NODE 如果保存 position 项，则也可通过它直接得到位置信息，否则需要通过 EDGE 间接得到；而对 FACE 则不能直接得到 FACE 的几何特征信息。

为了获得 NODE 和 FACE 的几何特征，需要根据拓扑规则创建两个函数：get_node()和 get_face()，它们函数可以通过 Oracle 的过程语言 PL/SQL 来实现，并保存到服务器中。这里给出函数的声明格式：

```
CREATE OR REPLACE FUNCTION get_node
(topolayer VARCHAR2(32), nodeid NUMBER)
RETURN mdsys.sdo_geometry;
CREATE OR REPLACE FUNCTION get_face
(topolayer VARCHAR2(32), faceid NUMBER)
RETURN mdsys.sdo_geometry;
```

4.3 拓扑图层的索引

为了提高空间数据的检索效率，必须分别对 NODE、EDGE、FACE 创建空间索引。对于 node 和 edge，它们的空间几何信息直接存贮在表的 SDO_GEOMETRY 类型的字段中，所以直接针对该字段来建索引。Oracle Spatial 支持两种类型的空间索引：四叉树索引和 R 树索引。如可以建立 EDGE 的 R 树索引：

```
CREATE INDEX rtindex_edge ON
```

(下转第88页)

表3 m[i][j]

	j:	1	2	3	4	5	6
i:	1	0	160000	208000	217000	295200	8952000
	2		0	80000	170000	952000	3952000
	3			0	90000	872000	3872000
	4				0	72000	2472000
	5					0	2400000
	6						0

例如在计算 m[1][5]时，依递归式有

$$m[1][5]=\min\begin{cases} m[1][1]+m[2][5]+N[1][1]*N[2][5]=2952000 \\ m[1][2]+m[3][5]+N[1][2]*N[3][5]=3032000 \\ m[1][3]+m[4][5]+N[1][3]*N[4][5]=3752000 \\ m[1][4]+m[5][5]+N[1][4]*N[5][5]=3770000 \end{cases}=2952000$$

且 k=1，因此 s[1][5]=1。

(4)构造最优解。算法 Dynamicjoin 只是计算出了最优值，并未给出最优解，也就是说，我们只得出要合并所给的集合链所需的最小计算量，还需进一步确定按什么次序来合并才能达到此最小计算量。算法 Dynamicjoin 已记录了构造一个最优解所需的全部信息，s[i][j]中的值 k 表示合并集合链 T[i:j]的最佳方式应在集合 Tk 和 Tk+1之间断开，即合并次序为(T[i:k])(T[k+1:j])。因此，从数组 s 中记录的值可知合并 T[1:n]的最优方式为(T[1:s[1][n]](T[s[1][n]+1:n])。而 T[1:s[1][n]]的最优合并方式为(T[1:s[1][s[1][n]]])(T[s[1][s[1][n]]+1:s[1][n]]；同理可确定 T[s[1][n]+1:n]的最优合并方式为在 s[s[1][n]+1][n]处断开……照此递推，最终可得集合链 T[1:n]的最优合并次序，即构造出问题的一个最优解。

下面的算法 Traceback 按算法 Dynamicjoin 计算出的断点矩阵 s 的值输出合并集合链 T[1:n]的最优次序。

```
void traceback(int i,int j,int * *s)
{
  if(i==j)return;
  Traceback(i,s[i][j],s);
  Traceback(s[i][j]+1,j,s);
  cout<<"Multiply T"<<i<<","<(s[i][j];
  cout<<"and T"<<(s[i][j]+1)<<",")
  <<endl
}
```

要输出 T[1:n]的最优合并次序只要调用 Traceback(1,n,s)即可。对于上例，通过调用 Traceback(1,6,s)，得到最优合并次序为：$(T_1(T_2(T_3(T_4T_5))))T_6$。

(5)算法 Dynamicjoin 复杂性分析。本算法的主要计算量取决于程序中对 r,i 和 k 的三重循环，循环体内的计算量为 O(1)，而三重循环的总次数为 $O(n^3)$，因此该算法的计算时间上界为 $O(n^3)$，算法占用的空间为 $O(n^2)$。由此可见，动态规划算法比穷举搜索法要有效得多。

结束语 随着 XML 作为 Internet 上数据表示和交换标准的出现，如何高效地进行 XML 数据的查询已变得越来越重要，本文介绍的分解合并方法，在处理有规则路径表示的 XML 数据的查询时，与传统的处理方法相比，效率有很大提高；在将动态规划算法用于中间结果集合的合并处理后，又在很大程度上加快了查询的处理速度，提高了查询效率。

参 考 文 献

1 Bray T, et al. Extensible markup language (XML) 1.0 second edition W3C recommendation: [Technical Report REC-xml-20001006]. World Wide Web Consortium, Oct. 2000
2 Abiteboul S, et al. The Lorel query language for semistructured data. International Journal on DigitalLibraries, 1997, 1(1): 68～88
3 Deutsch A, et al. A query language for XML. In: Proc. of the 8th Intl. World Wide Web Conf. Toronto, Canada, May 1999. 77～91
4 Ceri S, et al. XML-GL: A graphical language for querying and restructuring XML documents. In: Proc. of the 8th Intl. World Wide Web Conf. Toronto, Canada, may 1999. 93～109
5 Chamberlin D, et al. Xquery: A Query Language for XML W3C working draft: [Technical Report WD-xquery-20010215]. World Wide Web Consortium, Feb. 2001
6 Li Quanzhong, Moon B. Indexing and Querying XML Data for Regular Path Expressions. In: Proc. of the 27th VLDB Conf. Roma, Italy, 2001
7 Dietz P F. Maintaining order in a linked list. In: Proc. of the Fourteenth Annual ACM Symposium on Theory of Computing, San Francisco, California, May 1982. 122～127
8 McHugh J, Widom J. Query optimization for XML. In: Proc. of the 25th VLDB Conf. Edinburgh, Scotland, Sep. 1999. 315～326

(上接第79页)

```
edge_table_name (edge_info_column.geometry) INDEXTYPE IS
    MDSYS.SPATIAL_INDEX;
CREATE INDEX rtindex_node ON
node_table_name (node_info_column.position) INDEXTYPE IS
    MDSYS.SPATIAL_INDEX;
```

而对于 FACE，实际上并不存在一个 SDO_GEOMETRY 类型的字段。如果要对它建立索引，需要用到 Oracle Spatial 9i 的一个新功能，即建立基于函数的索引[1]，针对前面定义的 get_face()函数返回的 SDO_GEOMETRY 类型来建立空间索引：

```
CREATE INDEX rtindex_face ON face_table_name(get_face
(face_table_name, edge_id_column)) INDEXTYPE IS
MDSYS.SPATIAL_INDEX;
```

需要说明的是，如果要建立空间索引，Oracle Spatial 要求用于建索引的 sdo_geometry 字段在 user_sdo_geom_metadata 表中注册。所以，必须在该表中插入新的记录：

```
INSERT INTO user_sdo_geom_metadata VALUES (
edge_table_name, '
edge_info_column.geometry',
MDSYS.SDO_DIM_ARRAY (MDSYS.SDO_DIM_ELEMENT
('x', 110, 120, 0.1), MDSYS.SDO_DIM_ELEMENT('y', 30,
40, 0.1)), 8307);
```

结语 本文提出了在 Oracle 下进行拓扑管理的框架，它实现了在大型的 RDBMS 中存贮和管理空间拓扑的基本的模型。这种方案的基本思路就是使用 Oracle 提供的面向对象能力来定义拓扑对象，使用元数据表来建立拓扑相关表之间的关联，使用过程语言 PL/SQL 在服务器端实现基本的空间查询分析功能，同时，在必要的时候建立视图来提供更好的概念表达。

基于该模型，只需使用简单的 SQL 语句就可以进行常规的一些空间查询、分析，而对于该模型不同直接提供的空间处理能力，比如最短路径分析，可以通过编写 PL/SQL 过程或者使用 Oracle 的服务器端的 Java 扩展功能来实现，这个过程实际上就是将算法移植到 Oracle 服务器中的过程。

作为一个基本的框架，本文中没有提及在数据更新的情况下，如何维护拓扑关系一致性的问题。这也是我们以后需要研究的重点内容。从技术上讲，通过事务处理、触发器机制以及 Oracle 对 Java 语言的强大支持能力，可以在服务器端实现数据更新和拓扑关系维护。我们将继续在这方面进行更进一步的探索。

本文中提出的实现模型也可以扩展到其他的一些 RDBMS 平台上，比如 Informix、DB2等。

参 考 文 献

1 Oracle Corporation. Oracle9i SQL Reference Release 2 (9.2), March2002
2 Oracle Corporation. Oracle Spaital User's Guide and Reference Release 9.2, March2002
3 Watson P. Topology and ORDBMS technology. White Paper of Laser Scan Ltd, 2002
4 黄杏元，马劲松，汤勤. 地理信息系统概论(修订版). 北京：高等教育出版社，2001