GML 문서 저장을 위한 저장 스키마 및 하부 저장 관리자의 설계 및 구현

장 재 우[†] · 김 영 국[††] · 김 영 진[†††]

요 약

GML은 OGC(OpenGIS Consortium)에서 공간지리정보의 저장 및 전송을 위한 인코딩 표준으로 제안한 마크업 언어이다. 일반적인 공간 네트워크 데이터베이스에서 GML 지원을 위한 연구는 GML 문서의 파싱, GML 문서의 저장, GML 문서의 질의어로 분류된다. 이러한 3가지 주제 가운데 GML 문서 저장에 관한 연구는 효율적인 GML 문서 검색을 위해 필수적인 연구이다. 그러나 기존 XML 문서의 저장 스키마를 다루는 연구는 다수인 데 반해, GML 문서의 저장 스키마에 관한 연구는 거의 전무한 형편이다. 또한 기존 XML 문서 저장 스키마는 공간지리정보 저장에 적합하지 않다. 따라서 본 논문에서는 기존의 XML저장방식의 단점인 많은 중복데이터 저장, 엘리먼트를 얻기 위해 여러 테이블을 탐색해야 하는 단점을 보완하는 GML 문서의 효율적 저장을 위한 3가지의 저장 스키마를 제안한다. 아울러 제안하는 저장 스키마에 적합한 GML 지원 하부 저장 관리자를 설계 및 구현한다.

키워드 : 공간 네트워크데이터베이스, GML, 저장 스키마, 하부 저장 관리자

Design and Implementation of Storage Schema and Low-level Storage Manager for GML Documents

Jae-Woo Chang[†] · Young-Guk Kim[††] · Young-Jin Kim[†††]

ABSTRACT

GML is a markup language presented as exchange standard for geographic information by the OGC(Open GIS Consortium). In spatial network databases, researches for supporting GML(Geographic Markup Language) can be divided into the parsing, the storing and the retrieval of GML documents. Among them, the study on the storage of GML documents is essential for their efficient retrieval. However there is little research on the storing of GML documents whereas there have been a lot of researches on the storing of XML documents. Because the storage schema designed for XML documents are not appropriate for geographic information, we, in this paper, propose three storage schema for efficiently storing GML documents including geographic information in order to solve the problem that the XML storage schema store duplicate data and need to search many tables for obtaining elements. In addition, we design and implement a low-level storage manager which can store GML documents using the proposed GML storage schema.

Key Words : Spatial Network Database, GML, Storage Schema, Low-level Storage Manager

1. 서 론

이동통신기술의 발달로 휴대폰, PDA등 휴대용 단말기의 위치를 추적하여, 위치와 관련된 정보를 제공하는 LBS(location based service)에 대한 관심이 증가하고 있다. LBS에는 네트워크를 이용한 표준화된 서비스(3GPP), 위치정보 제공 또는 위치정보에 의해 정보를 제공하는 응용 소프트웨어 서비스(OGC), 사용자가 그들의 위치에 대한 서비스를 받도록 하는 것(FCC) 등이 중요한 영역으로 연구되고 있다. 또한 통신과 정보처리를 통합한 기술을 자동차에 제공하는 텔레메틱스(telemetics) 역시 LBS의 한 영역으로 자리 잡고 있다.

이런 LBS나 텔레메틱스 기술의 핵심은 공통적으로 지리정보를 이용한 서비스를 필요로 한다. OGC(Open GIS Consortium)는 네트워크, 응용 혹은 플랫폼의 형식에 관계없이 지리정보의 교환 표준으로 GML(Geographic Markup Language)을 채택했다[1]. GML은 피처(Feature)라고 불리는 지리적인 타입을 통해서 지리정보를 표현하고 있으며, 공간적인 피처와 비공간적인 피처를 포함한 지리정보를 전달하거나 저장하기 위해서 XML형태로 작성한다[2]. GML의 제안자인 Galdos 사는 GML을 지리정보 교환 표준으로 선택할 때 가질 수 있는 다양한 장점을 제시하고 있다[3]. 일

† 종신회원 : 전북대학교 컴퓨터공학과 교수
†† 정 회 원 : PENTAX VOICEWARE 연구원
††† 준 회 원 : 전북대학교 컴퓨터공학과 석사과정
논문접수 : 2006년 11월 22일, 심사완료 : 2007년 7월 13일

반적인 공간 데이터베이스에서 GML 지원을 위한 연구는 크게 3가지로 요약된다. GML 문서의 파싱(parsing), GML 문서의 저장, GML 질의어 등이다.

이러한 3가지 주제 가운데 GML 문서 저장에 관한 연구는·효율적인 GML 문서 검색을 위해 필수적인 연구이다. 그러나 기존 XML 문서의 저장 스키마를 다루는 연구는 다수인 데 반해, GML 문서의 저장 스키마에 관한 연구는 거의 전무한 형편이다. 효율적인 GML 문서 스키마를 설계하기 위해서는 새로운 도로나 POI(point of interest)가 생겼을 경우, 기존 문서나 저장 스키마에 대한 변경 없이 새로운 정보들만 추가적으로 저장해야 한다. 이를 위해서는 모델사상 기법이 적합하며, 기존의 가장 효율적인 기법은 XParent이다. 그러나 XParent는 공간 지리정보들의 표현의 한계성, 테이블 간의 데이터의 중복, 다수 테이블에 따른 검색의 비효율성의 문제점을 가진다. 따라서 논문에서는 XParent의 문제들을 해결하기 위해서 테이블 간의 결합을 고려하여 GML 문서의 효율적 저장을 위한 3가지 저장 스키마, 즉, GParent #1, GParent #2, GParent #3를 제안한다. 첫째, GParent #1는 데이터의 중복과 검색 시 별도의 Data 테이블의 검색을 줄이기 위해, Data2 테이블과 Element 테이블을 결합하여 하나의 테이블로 표현하는 스키마이다. 둘째, GParent #2는 데이터의 중복과 검색 시 별도의 DataPath 테이블의 검색을 줄이기 위해, Element 테이블과 DataPath 테이블의 결합하여 하나의 테이블로 표현하는 스키마이다. 셋째, GParent #3는 데이터의 중복과 검색 시 별도의 테이블들의 검색을 줄이기 위해, Data 테이블, Element 테이블과 DataPath 테이블을 결합하여 하나의 테이블로 표현하는 스키마이다. 본 논문에서는 아울러 제안하는 저장 스키마를 바탕으로, GML 문서를 저장하기 위한 GML 지원 하부 저장 관리자를 설계 및 구현한다. 성능평가를 통해서 제안한 GParent 방법이 삽입시간 및 저장 공간 사용량에서 기존의 XParent 방법보다 우수함을 나타내고, GML 문서의 삽입시간이 중요한 응용에는 GParent #2 저장 스키마가 최적이고, GML 문서의 검색시간이 중요한 응용에는 GParent #3 저장 스키마가 우수함을 제시한다. 또한 GML 지원 하부 저장 관리자의 성능비교를 통해 GML 문서의 삽입과 검색 시간에 있어서 상용 DBMS인 Oracle보다 우수함을 나타낸다.

2. 관련 연구

2.1 GML(geography markup language)

XML은 차세대 웹 문서의 표준으로 정착되어 다양한 분야의 문서들이 XML을 기반으로 생성되고 있다. 이에 따라 OGC(OpenGIS Consortium)에서는 XML을 기반으로 지리정보의 저장 및 전송을 위한 인코딩 표준으로 GML[1]을 제안하였다. 공간정보와 위치정보를 포함하는 기술들의 상호운용성(interoperability)에 목적을 두고 전 세계적으로 수백개가 넘는 기업, 정부기관, 대학들이 참여하는 컨소시움으로 만들어진 OGC는 개방형 지리정보시스템 환경을 위해 지리 정보 데이터와 응용 프로그램간 표준 인터페이스를 제시하는 것을 목표로 2002년까지 GML 2.0을 발표한 이후, 2003월 초에 GML 3.0을 제안하였다. GML 2.0에서는 XML DTD를 기반으로 하는 프로파일을 제공하는 GML 1.0을 포함하며, 단순 피쳐 모델에 기반하여 지오메트리 스키마(geometry.xsd), 피쳐 스키마(feature.xsd), XLinks 스키마(xlinks.xsd)의 세 가지 기본 XML 스키마를 정의하고 있다. 여기서 피쳐 스키마는 일반적인 피쳐-특성(feature-property) 모델을 정의하며, 지오메트리 스키마는 상세한 지오메트리 컴포넌트들을 나타낸다. 마지막으로 XLinks 스키마는 링크 기능을 구현하는 데 사용될 XLink 속성들을 지원한다. GML은 점(point), 연속선(linestring), 선형링(linearring), 다각형(polygon), 다중점(multipoint), 다중연속선(multilinestring), 다중다각형(multipolygon), 다중지오메트리(multigeometry) 클래스에 대한 지오메트리 엘리먼트와 좌표를 표현하기 위한 좌표(coordinate) 엘리먼트와 범위(extent)를 정의하기 위한 상자(box) 엘리먼트를 추가적으로 제공한다. GML 3.0은 기존의 GML과의 호환성을 제공하며, 모듈화, 복합 지오메트리(Geometry), 시공간 참조시스템, 위상, 메타데이터, 그리드 데이터, 측정 단위 등을 추가하였다. 특히 GML 3.0에서는 동적 객체 스키마를 통해 위치, 시간, 그리고 위상 개념 등을 포함함으로써 위치기반서비스에서 효과적으로 관리되어야 하는 이동 객체의 정보를 표현하는 데 활용될 수 있다.

2.2 XML / GML 저장 방식

일반적으로 관계형 데이터베이스에 XML/GML 문서를 저장하기 위한 연구는 모델 사상(model mapping)접근과 구조 사상(structure mapping)접근으로 분류할 수 있다[4, 5, 6]. 모델 사상기법은 정해진 데이터베이스 스키마를 이용하여 DTD나 스키마 문서 없이 XML/GML문서를 저장하는 기법이다. 즉, 모델 사상기법은 XML문서를 데이터 모델로 변환하는 과정에서 XML의 구조적 특징과 내용을 문서 독립적으로 모델링하여 저장하지 않는다.따라서 XML문서 구조의 갱신이 발생할 경우, 이에 따른 데이터베이스 구조정보가 수정되지 않아도 되는 장점이 존재한다. 모델 사상기법에는 간선(edge)기반과 정점(node)기반 기법으로 나눌 수 있다. 간선기반 기법은 기본적으로 XML문서를 표현한 그래프 상에서의 모든 간선들에 대한 정보를 간선(edge)라 불리는 하나의 테이블에 저장하는 방식이다. 이런 방식의 대표적인 시스템은 Monet[7]이다. Monet방식은 간선(edge)테이블을 가능한 레이블 경로(Label-Path)를 기반으로 분할하여 유일한 레이블 경로마다 하나의 테이블을 생성하는 방식이다. 한편, 정점(node)기반 기법은 XML문서를 표현한 그래프 상에서의 모든 노드들에 대한 정보를 가지고, 경로(path)정보, 노드(node)정보, 값(value)정보 등을 따로 분할하여 테이블로 저장하는 방식이다. 이를 대표하는 시스템으로는 XRel[8] 과 XParent[9] 등이 존재한다. XRel은 XML문서의 데이터를 Path, Element, Text, Attribute 네 개의 테이블에 저장한다. XParent는 XRel를 보다 성능을 개선시킨 방법으로, LabelPath, DataPath, Element, Data의 네 개의 기본 테이블이 있으며, 상위 Element로의

쉬운 접근을 위해서 추가로 Ancestor 테이블이 존재한다.

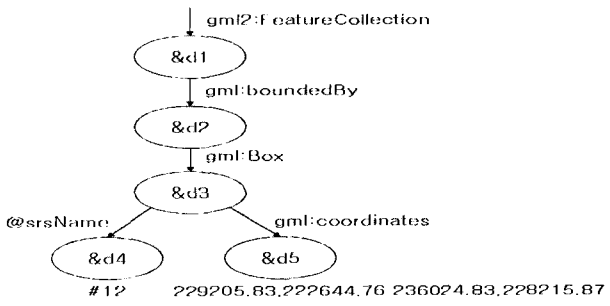
한편 XML/GML문서를 저장하는 또 다른 방법은 XML/GML문서의 구조 정보를 표현하는 DTD 또는 스키마를 기반으로 관계형 데이터베이스의 스키마를 정의하고 정의된 테이블에 XML/GML문서를 저장하는 구조 사상(structure mapping)방법이다. 구조 사상 방법은 XML/GML 스키마를 기반으로 생성된 관계형 데이터베이스(relational database)의 스키마(schema)에 따라 저장이 이루어지기 때문에, 스키마들을 만족하는 XML/GML문서들이 많이 존재하거나 스키마 생성에 사용된 스키마의 수정이 빈번하게 발생하지 않는 경우에 적합하다. 대표적인 연구로는 shared inlining/hybrid inlining 기법[10]과 비용 기반 접근 방식인 LegoDB[11]가 있다. inlining 기법들은 일반적으로 분석하고자 하는 DTD의 내용을 단순화(simplification)시킨 DTD그래프를 기반으로 관계형 데이터베이스의 스키마를 생성한다. 아울러 비용 기반(Cost-based)방식인 LegoDB는 관계형 데이터베이스의 스키마를 생성하기 위해 주어진 GML스키마의 의미(Semantics)를 벗어나지 않는 범위에서 다른 저장 구조를 유도하는 P-스키마를 생성하고, 스키마에 따른 비용을 계산하여 최소 비용을 가지는 스키마를 찾아내는 방법이다.

2.3 XParent[9]

모델사상기법의 여러 가지 방법 중 XParent 방법이 가장 발전된 형태이며 제일 우수하다고 알려져 있다[12]. XParent의 스키마는 5개의 테이블, 즉 LabelPath(PathID, Length, Path), Element(PathID, Ordinal, DataID), Data(PathID, DataID, Value), DataPath(Parent Data ID, Child Data ID), Ancestor (DataID, Ancestor Data ID, Level) 로 이루어져 있다. 여기서 밑줄친 항목은 Index 가 구성된 것을 나타낸다. 즉, LabelPath 테이블은 PathID, Path 항목이 Index로 구성이 되어 있고 Element 테이블은 PathID, DataID 항목이, DataPath 테이블은 Parent Data ID 항목이, Data 테이블은 모든 항목인 PathID, DataID, Value 항목이 Index로 구성되어 있다. 마지막으로 Ancestor 테이블은 모든 항목인 DataID, Ancestor Data ID, Level 항목이 Index로 구성된다. 각 테이블의 의미하는 바는 다음과 같다. 먼저, LabelPath 테이블은 XML 문서의 모든 Path 정보가 루트로부터 모두 포함이 되어 있다. 각각의 Path 하나에 고유한 pathID를 가지며 루트로부터 단계를 length로 표현하며 루트 자신은 length 1을 가진다. 둘째, Element 테이블은 XML 문서에서 모든 element와 path 와의

```
<gml2:FeatureCollection>
  <gml:boundedBy>
    <gml:Box srsName="#12"> <gml:coordinates>229205.83,222644.76
     236024.83,228215.87</gml:coordinates>
    </gml:Box>
  </gml:boundedBy>
</gml2:FeatureCollection>
```

(a) GML 문서의 예시



(b) 예시된 GML 문서의 Data Graph

(그림 1) GML 문서 및 문서의 Data Graph

PathID	Length	Path
1	1	gml2 : FeatureCollection
2	2	gml2 : FeatureCollection/gml:boundedBy
3	3	gml2 : FeatureCollection/gml:boundedBy/gml:Box
4	4	gml2 : FeatureCollection/gml:boundedBy/gml:Box/@ srsName
5	4	gml2 : FeatureCollection/gml:boundedBy/gml:Box/gml:coordinates

(a) Label Path 테이블

PathID	DataID	Value
4	&d4	#12
5	&d5	229205.83,222644.76 236024.83,228215.87

(b) Data 테이블

PathID	Ordinal	DataID
1	1	&d1
2	1	&d2
3	1	&d3
4	1	&d4
5	1	&d5

(c) Element 테이블

Parent ID	Child ID
&d1	&d2
&d2	&d3
&d3	&d4
&d3	&d5

(d) Data Path 테이블

DID	AID	Level
&d2	&d1	1
&d3	&d1	2
&d4	&d1	3
&d5	&d1	3
&d3	&d2	1
&d4	&d2	2
&d5	&d2	2
&d4	&d3	1
&d5	&d3	1

(e) Ancestor 테이블

(그림 2) XParent의 5개의 테이블

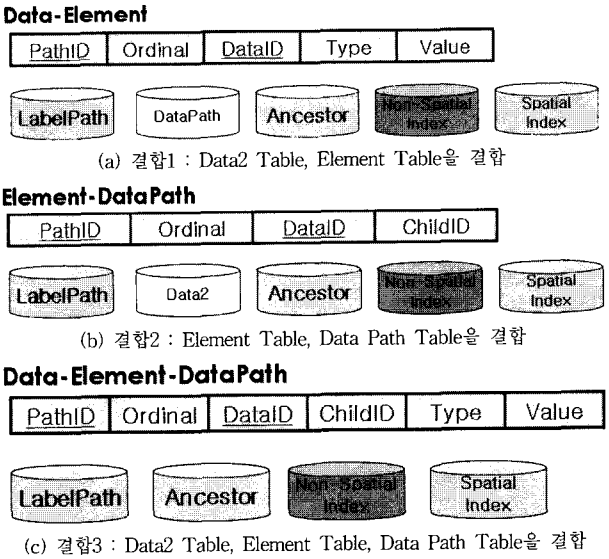
관계를 나타내는 테이블로 각각의 element는 고유한 DataID를 가지며, 어떤 path와 부합하는지 표현하기 위해 PathID가 있으며, 같은 path 일 경우 순위를 나타내는 order가 구성되어 있다. 셋째, DataPath 테이블은 부모-자식 엘리먼트와의 관계를 나타내는 테이블로 각각의 엘리먼트에 자식 엘리먼트가 존재할 때 Parent Data ID와 Child Data ID로 구성되어 있다. 넷째, Data 테이블은 각각의 엘리먼트에 Data 정보를 담고 있는 테이블로 PathID, DataID 와 실제 Data를 담고 있는 Value로 구성된다. 마지막으로, Ancestor 테이블은 상위 엘리먼트와의 관계를 나타내는 테이블로 각각의 엘리먼트의 모든 조상들을 Level 별로 표현하고 있다. 한편 XParent의 테이블의 예시를 살펴보면, (그림 1(a))와 같은 GML 문서가 있을 때, (그림 1(b))는 그 문서를 Data Graph로서 표현한 것이다. (그림 2)는 (그림 1(b))의 Data Graph를 토대로 XParent의 5개의 테이블로 표현한 것이다.

3. GML 문서를 위한 저장 스키마의 설계

3.1 설계 시 고려사항

GML 문서 저장을 위한 스키마 설계 시 고려사항은 다음과 같다. 첫째, GML문서는 공간 지리정보를 포함하여 다양한 형태로 변환 및 사용이 가능하다. 따라서 이미지 도형처리가 가능한 SVG(Scalable Vector Graphics)형태로 변환하고 필요한 데이터를 서버(Server)독립적으로 처리할 수 있어야 한다. 또한 주어진 지점에서 반경 10Km 내의 주유소를 검색하여 조건을 만족하는 GML 문서를 클라이언트(client)에 보내면, 클라이언트는 GML 문서를 사용하여 브라우징이 가능해야 하고 GML 문서를 원문 형식으로 보낼 수 있어야 한다. 둘째, 본 논문에서 적용하고자 하는 LBS나 텔레매틱스 응용에서는 새로운 도로나 POI(point of interest)가 생겼을 경우, 기존 문서나 저장 스키마에 대한 변경 없이 새로운 정보들만 추가적으로 저장하는 것이 요구된다. 또한 GML 문서 스키마의 국제 표준은 OGC(Open Geospatial Consortium)에서 지속적으로 진행 중이다. (현재까지 개발된 GML 문서의 스키마 버전은 3.1.1이다.) 따라서 새로운 버전의 GML 스키마로 작성된 문서를 추가하고자 할 경우, 기존 문서나 저장 스키마의 구조적 변경 없이 새로운 문서를 추가하는 방법이 필요하다. 한편 XML/GML 저장기법 가운데 위의 2가지의 고려사항을 만족하는 기법은 모델사상 기법이다. 구조사상 기법은 스키마에 따른 비용을 계산하여 최소 비용을 가지는 스키마를 찾아내는 효율적인 기법이지만, GML 문서 저장 후 원본 GML 문서를 복원할 수 없는 단점을 지니고 있다. 따라서 새롭게 추가되는 GML 문서가 기존의 문서와 스키마 버전 또는 구조가 다를 경우, 추가되는 문서의 수가 매우 적어도 기존에 저장된 다수의 문서들을 그들의 스키마를 변경하여 새롭게 다시 저장해야 하는 단점을 지닌다.

한편 모델 사상기법 가운데 가장 효율적인 기법은 XParent이다. 그러나 XParent 는 다음과 같은 문제점을 지니고 있다. 첫째, 공간 지리정보들의 표현의 한계성 및 자료 검색의 한계성을 지니고 있다. 둘째, 테이블 간의 데이터의 중복으로 인한 저장 공간(disk)의 낭비가 존재한다. 마지막으로, 테이블 수가 많아 하나의 엘리먼트를 찾기 위해서는 여러 개의 테이블을 검색해야 한다. 둘째, XParent 저장 스키마는 일반 텍스트로만 이루어진 XML 문서에 맞추어서 저장하도록 설계된 스키마이다. 따라서 GML 문서에서 공간지리정보를 포함한 문서를 저장하기 위해서는, GML 문서에서 공간지리정보에 대한 표현 및 색인을 제공해야만 한다. 그러므로 Data 테이블에 대한 확장이 필요하다. Data 테이블에 *"Type"* 항목을 추가해서 Data 테이블 내에서의 Data Type을 구분하고, 공간지리정보 (Spatial Data)와 비공간지리정보 (Non-Spatial Data)에 대한 색인을 별도로 제공해야 한다. 확장된 Data 테이블을 Data2 테이블이라 하고, 추가된 색인을 Spatial Index, Non-Spatial Index라 지칭한다. Spatial Data 지원을 위한 XParent의 최소 확장과 추가된 Index는, LabelPath(PathID, Length, Path), Data2(PathID, DataID, Type, Value), Element (PathID, Ordinal, DataID), DataPath(Parent Data ID, Child Data ID), Ancestor(DataID, Ancestor Data ID, Level), Non-Spatial Index(Value, Type, DataID), Spatial Index(Value, Type, DataID) 이다. 여기서 bold 체는 확장되거나 추가된 테이블을 나타낸다. Non-Spatial Index와 Spatial Index는 *"Value"*를 키 값으로 가지며, 그 외의 항목인 Type과 DataID를 데이터 값으로 가진다. 이러한 확장을 통하여 XParent의 첫 번째 문제를 해결할 수 있다. 둘째, XParent의 Data · Element · DataPath 테이블의 데이터들이 독립적인 테이블로, 의미적으로는 각기 다른 내용을 담고 있지만 전체적인 관점으로 본다면 두 번째 문제인 데이터 중복의 문제가 발생한다. 셋째, 하나의 엘리먼트를 검색하기 위해서 Data · Element · DataPath 테이블을 모두 검색한 후에, 하위 엘리먼트가 존재하는지 데이터가 존재하는지 판단할 수 있기 때문에 3개의 테이블을 모두 검색해야 하는 문제가 발생한다. 이와 같은 두 가지 문제를 해결하기 위해서 테이블간의 결합을 고려한



(a) 결합1 : Data2 Table, Element Table을 결합

(b) 결합2 : Element Table, Data Path Table을 결합

(c) 결합3 : Data2 Table, Element Table, Data Path Table을 결합

(그림 3) 3가지 방법의 테이블 결합

다. XParent의 Data2 · Element 테이블은 공통적으로 동일한 PathID와 DataID를 사용하며, Data2 · Element · DataPath 테이블은 공통적으로 동일한 DataID를 사용한다. 따라서 (그림 3)에서와 같이 전체 5개의 테이블과 두 개의 Index 중 3가지 경우의 결합을 고려한다. 즉, Data2 · Element 테이블의 결합(결합 1), Element · DataPath 테이블의 결합(2), Data2 · Element · DataPath 테이블의 결합(3)이다. (그림 3)은 3개의 결합을 나타낸다. 따라서 본 논문에서는 결합 1, 2, 3 에 각각 해당하는 GML 문서를 위한 새로운 저장 스키마 GParent #1, GParent #2, GParent #3을 제안한다.

3.2 제안하는 저장 스키마 1

XML/GML 문서는 엘리먼트에 데이터가 존재하는 경우가 있다. 하지만 XParent의 경우에는 엘리먼트에 포함된 데이터를 얻어오기 위해 Element 테이블과 Data2 테이블로부터 동일한 PathID와 DataID를 갖는 레코드를 검색하기 위해 조인(Join) 연산이 불가피하다. 따라서 이러한 점을 해결하기 위해 XParent의 Data2 테이블과 Element 테이블을 결합하여 하나의 테이블로 표현한다. 이러한 테이블을 Data-Element 테이블이라 하고, 또한 이렇게 확장된 저장 스키마를 GParent #1 이라고 명명한다. (그림 4)는 GParent #1의 전체 4개의 테이블과 추가된 Index를 나타낸다. Data-Element 테이블의 Index 값은 확장된 XParent의 Data2 테이블의 Index 값과 같은 PathID, DataID 항목으로 이루어져 있다. Data-Element 테이블을 사용한 GParent #1는 XParent에서 발생하는 자료의 중복현상을 없애고, 질의를 처리할 시에 Element 테이블과 Data 테이블과의 조인 연산 없이 그 결과를 바로 도출할 수 있다는 장점을 가지고 있다. 하지만 이 방법은 완전한 엘리먼트를 얻기 위해서 추가적으로 DataPath 테이블을 검색해야한다.

3.3 제안하는 저장 스키마 2

XML/GML 문서는 하나의 엘리먼트에 바로 데이터가 있는 경우도 있지만, 그렇지 않고 하위 엘리먼트가 존재할 가능성이 있다. XParent의 경우 이를 처리하기 위해서 DataPaht 테이블을 이용해서 하위에 어떤 엘리먼트가 존재하는지 그 DataID를 얻어온다. 그 얻어온 DataID를 이용하여 다시 Element 테이블을 검색한다. 이러한 이유로 하위 엘리먼트를 찾기 위해 DataPath 테이블과 Element 테이블과의 조인 연산은 불가피 하다. 따라서 효율적인 조인을 위해 XParent의 Element 테이블과 DataPath 테이블의 결합하여 하나의 테이블로 표현한다. 이 결합된 테이블을 Element-DataPath 테이블이라 하고, 또한 이렇게 확장된 저장 스키마를 GParent #2 라고 명명한다. (그림 5)는 GParent #2의 전체 4개의 테이블과 추가된 Index를 나타낸다. Element-DataPath 테이블의 Index 값은 XParent의 Element 테이블의 Index 값과 같은 PathID, DataID 항목으로 이루어져 있다. Element-DataPath 테이블을 사용한 GParent #2는 질의를 처리할 시에 Element 테이블에서 하위 엘리먼트가 존재하거나 존재하지 않거나 DataPath 테이블의 검색을 줄일 수 있다는 장점을 가지고 있다. 이러한 조인 연산을 줄여 하위 엘리먼트의 DataID를 보다 빠르게 검색할 수 있게 된다. 하지만 이 방법은 완전한 엘리먼트를 얻기 위하여 추가적으로 Data 테이블을 검색해야한다.

3.4 제안하는 저장 스키마 3

XParent는 어트리뷰트를 하위 엘리먼트의 한 종류로 보기 때문에, 하나의 엘리먼트가 하위 엘리먼트를 가지면서 동시에 데이터를 지닐 수 있다. 따라서 하위 엘리먼트가 존재해도 Data 테이블의 검색이 필요할 수 도 있다. 이러한 문제점을 해결하기 위해 XParent의 Data 테이블, Element 테이블과 DataPath 테이블을 결합하여 하나의 테이블로 표현한다. 이 테이블을 Data-Element-DataPath 테이블이라 하고, 이렇게 확장된 저장 스키마를 GParent #3 이라고 명명한다. 그림 6은 GParent #3의 전체 3개의 테이블과 추가된 Index 를 나타낸다. Data-Element-DataPath 테이블의 Index 값은 XParent의 Element 테이블의 Index 값과 같은 PathID, DataID 항목으로 이루어져 있다. Data-Element-DataPath 테이블을 사용한 GParent #3은 Xparent의 단점인 자료의 중복현상을 줄이고 질의 수행 시 하나의 엘리먼트의 검색을 위해서 최대 두 번의 조인 연산을 줄일 수 있다는 장점을 가지고 있다. 하지만 이 방법은 테이블에 삽입되는 레코드의 크기가 증가하면 테이블의 삽입/검색 성능의 저하를 초래할 수 있게 된다.

3.5 3가지 저장 스키마의 예시 및 장단점 비교

본 절에서는 제안한 저장스키마를 실제 GML 문서에 적용시켰을 때의 각 테이블들의 예를 제시한다. 적용할 GML 문서는 (그림 1)의 GML 문서를 사용한다. (그림 7(a))는 XParent의 문제점을 해결하기 위해서 제시한 Data2 테이블

```
LabelPath(PathID, Length, Path)
Data-Element(PathID, Ordianl, DataID, Type, Value)
DataPath(Parent Data ID, Child Data ID)
Ancestor(DataID, Ancestor Data ID, Level)
Non-Spatial Index(Value, Type, DataID)
Spatial Index(Value, Type, DataID)
```

(그림 4) GParent #1 저장스키마

```
LabelPath(PathID, Length, Path)
Data2(PathID, DataID, Type, Value)
Element-DataPath(PathID, Ordinal, DataID, Child Data ID)
Ancestor(DataID, Ancestor Data ID, Level)
Non-Spatial Index(Value, Type, DataID)
Spatial Index(Value, Type, DataID)
```

(그림 5) GParent #2 저장스키마

```
LabelPath(PathID, Length, Path)
Data-Element-DataPath
(PathID, Ordianl, DataID, Type, Value, Child Data ID)
Ancestor(DataID, Ancestor Data ID, Level)
Non-Spatial Index(Value, Type, DataID)
Spatial Index(Value, Type, DataID)
```

(그림 6) GParent #3 저장스키마

과 Element 테이블을 결합한 GParent#1의 Data-Element 테이블의 예시이다. 그 외의 LabelPath, DataPath, Ancestor 테이블은 확장하지 않았으므로 (그림 2)의 (a), (d), (e)와 동일하며, Non-Spatial Index와 Spatial Index는 (그림 2)의 (b), (c)와 동일하다. (그림 7(b))는 XParent의 문제점을 해결하기 위해 제시한 GParent #2의 Element-DataPath 테이블의 예시이다. 그 외의 LabelPath, Ancestor 테이블은 확장하지 않았으므로 (그림 2)의 (a), (e)와 동일하며, Data2 테이블, Non-Spatial Index와 Spatial Index는 (그림 2)의 (a), (b), (c)와 동일하다. (그림 7(c))는 XParent의 문제점을 해결하기 위해서 제시한 Data2 테이블, Element 테이블과 DataPath 테이블을 결합한 GParent #3의 Data-Element-DataPath 테이블의 예시이다. 그 외의 LabelPath, Ancestor 테이블은 확장을 하지 않았으므로 (그림 2)의 (a), (e)와 동일하며, Non-Spatial Index와 Spatial Index는 (그림 2)의 (b), (c)와 동일하다.

마지막으로 기존 XParent 및 본 논문에서 제안하는 GParent #1, GParent #2, GParent #3 저장스키마의 장단점을 비교하면 <표 1>과 같다.

PathID	Ordinal	DataID	Type	Value
1	1	&d1	Null	
2	1	&d2	Null	
3	1	&d3	Null	
4	1	&d4	ATR	#12
5	1	&d5	Polygon	229205.83,222644.76 236024.83,228215.87

(a) GParent #1의 Data-Element 테이블

PathID	Ordinal	DataID	Child Data ID
1	1	&d1	&d2
2	1	&d2	&d3
3	1	&d3	&d4, &d5
4	1	&d4	
5	1	&d5	

(b) GParent #2의 Element-DataPath 테이블

PID	Ord	DID	Child Data ID	Type	Value
1	1	&d1	&d2	Null	
2	1	&d2	&d3	Null	
3	1	&d3	&d4, &d5	Null	
4	1	&d4		ATR	#12
5	1	&d5		Polygon	229205.83,222644.76 236024.83,228215.87

(c) GParent #3의 Data-Element-DataPath 테이블

(그림 7) GParent #1, GParent #2, GParent #3 예시 테이블

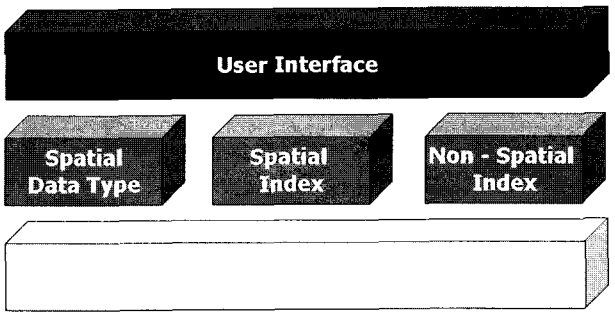
〈표 1〉 저장스키마의 장단점 비교

저장 스키마	장점	단점
XParent	테이블을 의미적으로 구분하여 쉽게 알아볼 수 있다.	데이터의 중복이 많고 검색시 여러개의 테이블을 참조해야한다.
GParent #1	데이터의 중복과 검색 시 별도의 Data 테이블의 검색을 줄임	DataPath 테이블을 검색해야 한다.
GParent #2	데이터의 중복과 검색시 별도의 DataPath 테이블의 검색을 줄임.	Data 테이블을 검색해야한다.
GParent #3	데이터의 중복과 검색시 별도의 테이블들의 검색을 줄임	하나의 레코드 크기가 증가하므로 삽입/검색시 성능 저하 현상이 발생할 수 있음.

4. GML 문서를 위한 하부 저장 관리자

본 절에서는 제안하는 저장 스키마를 바탕으로 GML 문서를 저장하기 위한 GML 지원 하부 저장 관리자를 설계 및 구현한다. 구현된 GML 지원 하부 저장 관리자는 총 5개의 모듈로 구성된다. 즉, storage manager, spatial data type manager, spatial index, non-spatial index, user interface로 이루어져 있으며, (그림 8)는 전체 구조를 나타낸다. 첫째, storage manager는 하부저장 시스템에서 최하위단으로 저장스키마에 따라 저장된 데이터의 저장/검색/수정/삭제 등을 담당한다. 아울러 트랜잭션(tansaction), 락킹(Locking), 로깅(Logging)을 제공하는 모듈이다. 둘째, spatial data type manager는 공간지리정보 타입의 지원을 위한 모듈이다. 일반적인 데이터 타입인 정수, 실수 타입을 확장하여 spatial data type인 point(s), polyline(s), polygon(s) 등을 지원하는 모듈이다. 셋째, spatial index는 공간지리정보의 색인을 지원하기 위한 모듈이다. 즉, Spatial Data Type Manager에서 다루는 공간 데이터에 대한 색인을 구성하는 모듈이다. 이를 통하여 공간지리정보의 검색/저장을 담당한다. 넷째, non spatial index 는 spatial data type을 제외한 non-spatial type의 데이터에 대한 색인을 구성하는 모듈이다. 즉, spatial data index 이외의 모든 테이블의 색인 구성을 담당한다. 마지막으로, user interface는 하부저장 시스템에서 최상위단에 위치하며, 하위 모듈을 관리하고 사용자와 하부저장구조와의 연결을 위한 인터페이스를 제공한다. 사용자는 이 모듈을 통해 원하는 GML 문서를 얻기 위해서 질의를 입력하면, 이 모듈은 하부 모듈들에 질의를 수행하도록 명령을 내린다. 아울러, 질의의 결과물인 GML 문서를 사용자에게 반환한다.

GML 문서를 위한 하부 저장 관리자의 구현 환경은 Windows Server 2003 운영체제 하에서, Storage Manager Module로 Berkeley DB 4.3.29, 그리고 컴파일러로서 Visual C++ 7.1을 사용한다. 아울러 구현된 시스템이 제공하는 API는 user interface API, 테이블 manger API, spatial data type API, RST API, Berkeley DB API 등이다. 첫째, user iterface API는 하위의 모든 API를 관리하며, 전체시스템의 최상위 interface를 제공한다. 즉, user interface API를 통해서 GML 문서의 저장/검색을 수행한다. 이러한 최상위 인터페이스를 사용함으로써, 전체 테이블 관리를 일관성 있게 유지할 수 있다. 둘째, GML 문서의 저장 시, 테이블 manager API를 호출하여 저장하고 이를 통해 GML 문서를 생성하여 사용자에게 반환한



(그림 8) 하부 저장 관리자의 전체 구조

다. 아울러 GML 문서의 검색 시, 테이블 Manager API 를 호출하여 검색을 수행한다. 셋째, Spatial Data Type API는 Spatial Data Type 모듈에 해당되며 Point(s), Polyline(s), Polygon(s) 클래스로 구성된다. 이 API를 통해서 모든 공간 데이터들에 대한 효율적인 관리를 수행한다. 넷째, 공간 데이터의 색인을 위해서는 RST API를 사용한다. 마지막으로, Berkeley DB API를 통해서 실제 Disk와 Memory에 반영되며, 이는 Sleepycat[13]사에서 제공하는 Berkeley DB C++ 라이브러리를 사용하였다. 한편, storage manager 모듈로 Berkeley DB를 사용하는 이유는 Berkeley DB가 오픈소스 임베디드 데이터베이스 라이브러리이며, 데이터베이스로서 갖추어야 할 동시성 제어, 트랜잭션(transaction) 복구(recovery) 기법을 제공하기 때문이다. Berkeley DB는 DBMS가 아닌 데이터베이스 라이브러리이기 때문에, 본 논문에서 설계한 storage manager 모듈로 적합하다고 할 수 있다.

5. 성능평가

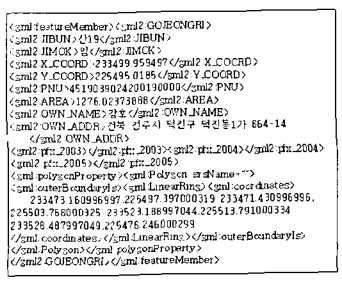
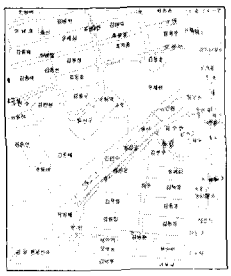
본 절에서는 저장스키마의 효율성을 성능평가하기 위해, 첫째로 기존 저장스키마인 XParent와 본 논문에서 제안한 3개의 저장스키마 즉, GParent #1, GParent #2, GParent #3의 효율성을 평가한다. 효율성에 대한 항목으로는 GML 문서의 삽입시간, 저장공간 사용량 및 엘리먼트당 검색시간을 측정한다. 둘째, 본 논문에서 제시한 GML 지원 하부 저장 관리자의 효율성을 평가하기 위해, 기존 상용 DBMS인 Oracle과 성능 비교를 수행한다. 효율성에 대한 항목으로는 GML 문서의 삽입시간, 엘리먼트당 검색 시간 및 Spatial Data에 대한 검색 시간을 측정한다. 성능평가는 Intel Xeon 3.0Ghz CPU(메모리 Ram 2GB)에서 동작하는 Windows Server 2003 운영체제 하에서, 컴파일러로 Visual C++ 7.1을 사용하여 수행하였다. 또한 Oracle은 공간 데이터를 처리할 수 있는 Oracle 9i Release 2(9.2.0.1)를 사용하였다. 입력 데이터는 두 종류의 데이터를 사용한다. 첫째, GML Document Type A는 (주)대경지리정보[14]에서 제공한 전라북도 남원시 덕과면 고정리의 데이터이고, GML Document Type B는 (주)Thinkware[15]에서 제공한 전라북도 내 도시 데이터이다. Type A 문서의 특징은 문서의 단계가 깊지 않으며, spatial Data 보다 non-spatial Data를 다소 많이 포함하고 있는 반면, Type B 문서의 특징은 문서의 단계가 깊으며, non-spatial data 보다 spatial data를 보다 많이 포함하고 있다. <표 2>는 본 실험에서 사용한 데이터를 나타낸다. (그림 9)는 실험에서 사용한 대표적인 데이터의 세부 문서내용과 그 문서를 이용한 시각적 지도 표현을 나타낸다. 첫 번째로 고정리 지역의 GML문서(gojungri.gml)는 Type A에 속하는 가장 큰 크기의 문서이고, 두 번째로 전주시 지역의 GML문서(jeonju.gml)는 Type B에 속하는 문서이며 그 이외의 문서는 두 대표문서와 동일한 구조를 가진다.

5.1 저장스키마의 성능평가

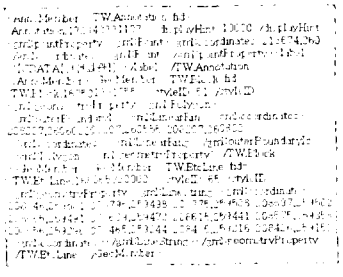
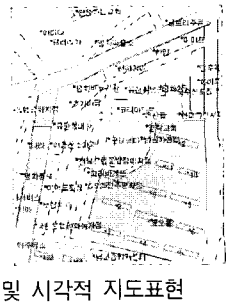
첫째, (그림 10)은 GML문서의 삽입 성능을 나타낸다. 기존 스키마인 XParent 가 가장 좋지 않은 성능을 보이며, 그 다음으로 GParent #1의 성능이 좋지 못하다. Type A 문서들의 경우, 입력된 GML 문서의 크기가 작으면 GParent #3의 성능이 가장 좋으며, GML 문서의 크기가 커질수록 GParent #2의 성능이 다른 스키마에 비해서 좋아진다. Type B 문서들의 경우, GParent #2의 성능이 가장 좋다. 그 이유는 GParent #1, GParent #2, GParent #3 저장스키마는 DataPath 테이블을 사용하지 않고, 개선된 Element-DataPath, Data-Element-DataPath

〈표 2〉 실험에서 사용한 데이터

문서구분	문서 이름	문서 크기	엘리먼트수 (공간자료포함수)	비고
Type A	river.gml	977KB	14,329(1,433)	하천정보 데이터
	text.gml	793KB	24,165(2,196)	POI 정보 데이터
	road.gml	1.86MB	34,539(3,454)	도로정보 데이터
	gojungri.gml	3.35MB	55,565(2,925)	토지 사용정보 데이터
	bldg.gml	3.00MB	72,717(6,060)	빌딩정보 데이터
	farm.gml	16.6MB	371,279(37,128)	농지정보 데이터
Type B	jeonju.gml	5.70MB	156,300(19,117)	전주시 데이터
	kunsan.gml	6.90MB	181,379(21,118)	군산시 데이터
	iksan.gml	7.59MB	186,776(21,919)	익산시 데이터

(그림 9-1) gojungri.gml의 문서내용 및 시각적 지도표현

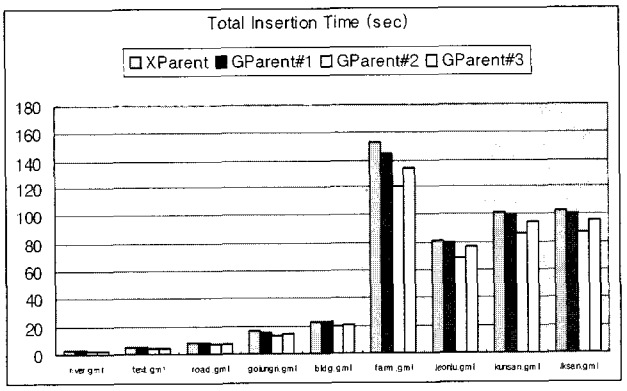
(그림 9-2) jeonju.gml의 문서내용 및 시각적 지도표현

테이블을 사용하기 때문이다. 또한, GML 문서가 커질 경우, GParent #3가 GParent #2에 비해 삽입 성능이 저하되는 이유는 Data-Element-DataPath 테이블의 전체 크기가 레코드 수가 증가할수록 커지기 때문이다.

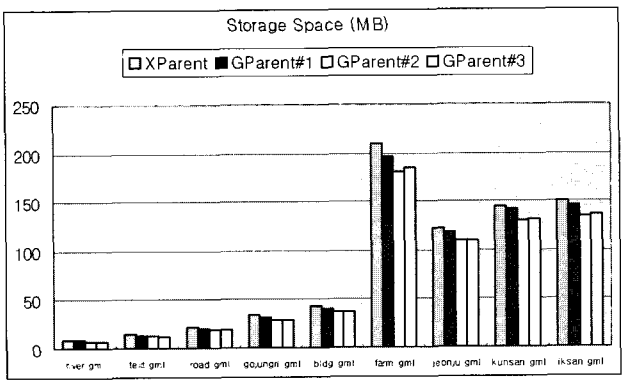
둘째, (그림 11)은 GML문서들의 저장 공간 사용량을 나타낸다. 앞서 4가지의 저장스키마중 자료의 중복이 제일 적게 설계된 스키마는 GParent #3이다. 그러나 GML 문서의 크기가 커질수록, 즉 엘리먼트의 수가 많아질수록, GParent #2가 저장 공간의 사용량이 적다. 그 이유는 Data-Element-DataPath 테이블의 전체 크기가 레코드수가 증가할수록 커지기 때문에, 저장 공간의 사용량이 많아진다. 따라서 GML 문서의 크기가 작다면 GParent #3 스키마가 효율적이며, GML 문서가 크다면 GParent #2 스키마를 사용하는 것이 효율적이다.

마지막으로 본 논문에서 제안한 저장스키마인 GParent #1, GParent #2, GParent #3의 검색 성능평가를 수행한다. GML을 검색하기 위한 모든 질의는 GML 문서에서의 Data ID를 얻기 전 과정과 얻은 후의 과정으로 나뉠 수 있는데, 예를 들면 Path만을 알고 있을 때에는 Label Path 테이블에서 Path를 가지고 검색하여 Path ID를 얻는다. 그 후 XParent의 경우에는 Element 테이블을, GParent #1의 경우에는 Data-Element 테이블을, GParent #2의 경우에는 Element-DataPath 테이블을, 그리고 GParent #3의 경우에는 Data-Element-DataPath 테이블을 검색하여 Data ID를 얻는다. 이렇게 얻어진 Data ID를 통해 GML 문서의 해당하는 엘리먼트를 얻는다. 마찬가지로 Non-Spatial 값이나 Spatial 값도 Non-Spatial Index와 Spatial Index의 검색을 통해 Data ID를 얻은 후, 해당하는 엘리먼트를 얻는다. 본 논문에서 비교하고자 하는 4개의 저장스키마의 가장 큰 차이점은 Data 테이블, Element 테이블, Data Path 테이블의 조합으로 이루어진 Data-Element, Element-DataPath, Data-Element-DataPath 이므로 DataID를 통한 검색을 통해 검색 성능을 평가한다. <표 2>에 소개된 문서들의 모든 DataID에 대해서 평균 검색시간을 측정한다. 예를 들면 가장 크기가 큰 farm.gml 문서는 총 371,279개의 엘리먼트를 가지므로 DataID = 1 ~ 371,279에 대한 평균 엘리먼트당 검색시간을 측정한다. DataID가 주어질 때 결과로 나온 시간을 검색된 GML 문서가 갖는 엘리먼트수로 나누어서 평균 엘리먼트당 검색시간을 측정한다. (그림 11)은 GML문서들의 엘리먼트당 평균 검색시간을 비교한 것이다. 4가지의 저장스키마중 기존 스키마인 XParent에서 가장 좋지 않는 성능을 보이며, Type A 문서들의 경우에는 GParent #2, Type B 문서들의 경우에는 GParent #1의 성능이 좋지 못하다. 그 이유는 Type A 문서들의 경우에는 문서의 깊이가 얕기 때문에, 하위 엘리먼트의 정보를 지니고 있지 않아서 Data 테이블의 빈번한 접근이 요구된다. 따라서 Element-DataPath 테이블 보다 Element-Data 테이블을 사용하는 것이 경제적이다. 반면 Type B 문서의 경우에는 문서의 깊이가 깊기 때문에, 하위 엘리먼트의 정보를 많이 포함함으로써 DataPath 테이블에 대한 빈번한 접근이 요구된다. 따라서 Element-Data 테이블 보다 Element-DataPath 테이블을 사용하는 것이 경제적이다. 문서의 Type에 상관없이 GParent #3의 경우에는 가장 좋은 성능을 보이고 있으며, XParent와 비교해서 최대 약 45%, 최소 약 63%의 성능 향상을 보인다.

결론적으로 성능평가를 통해서, 삽입시간과 저장 공간 사용량은 GParent #2 저장스키마가 우수하며, 검색시간은 GParent #3 저장스키마가 우수하다. GML 문서의 삽입시간이 중요한 응용이라면 GParent #2 저장스키마를 사용하고, GML 문서의 검색시간이 중요한 응용이라면 GParent #3 저장스키마를 사용하는 것이 적절하다.



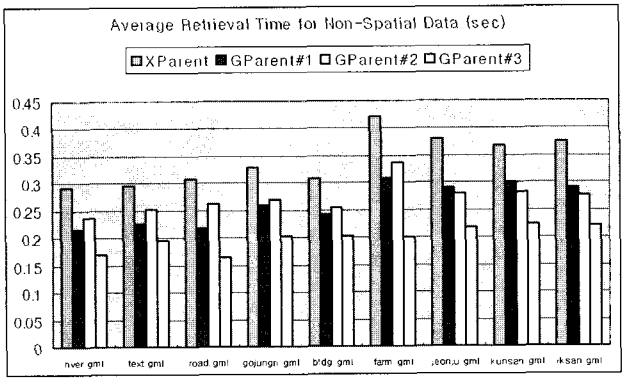
(그림 10) GML 문서들의 전체 삽입 성능



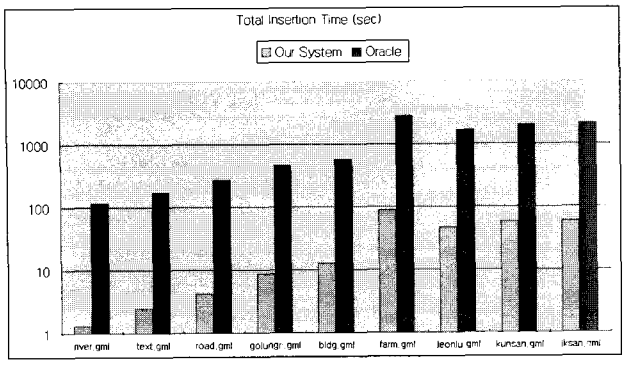
(그림 11) GML 문서들의 저장공간 사용량 비교

5.2 하부저장관리자의 성능평가

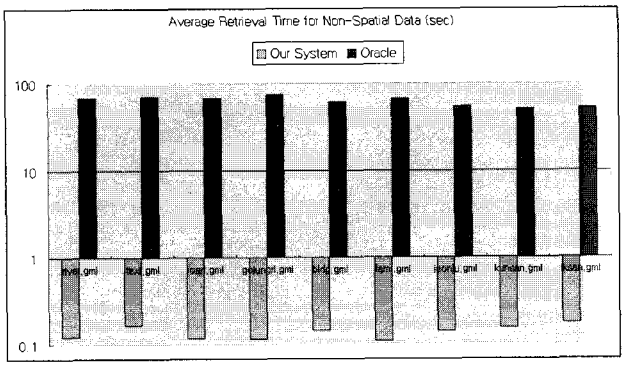
본 절에서는 제안하는 저장스키마중 가장 검색 효율성이



(그림 12) 엘리먼트당 평균 검색시간의 성능 비교



(그림 13) GML 문서들의 전체 삽입시간의 성능 비교



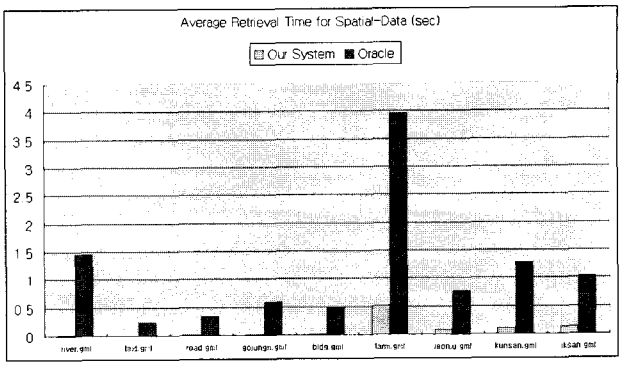
(그림 14) 엘리먼트당 평균 검색시간의 성능 비교

우수한 GParent #3 을 이용하여, 본 논문에서 제시한 GML 지원 하부저장관리자 및 공간 데이터의 저장이 가능한 대표적인 상업용 DBMS 인 Oracle 과 성능비교를 수행한다. 먼저 (그림 13)은 GML문서의 삽입시간 성능을 비교한 것이다.

데이터의 크기가 적은 경우, 즉 river.gml 의 경우 Oracle 은 저장시 약 116 초가 소요되고, 본 논문에서 제시한 GML 지원 하부 저장 관리자는 약 1.3 초가 소요된다. 아울러 데이터의 크기가 큰 경우, 즉 farm.gml의 경우 Oracle 은 저장을 위해 약 2700 초가 소요되고, 본 논문에서 제시한 GML 지원 하부 저장 관리자는 약 88초가 소요된다. GML 문서의 삽입의 경우, 최소 약 30배에서 최대 약 90배 까지 Oracle 에 비하여 성능이 우수함을 알 수 있다.

둘째, (그림 14)는 GML문서의 엘리먼트당 평균 검색시간의 성능을 비교한 것이다. 데이터의 크기가 적은 경우, 즉 river.gml의 경우 Oracle 은 엘리먼트당 평균 검색시간이 약 68초가 요구되나, 본 논문에서 제시한 GML지원 하부 저장 관리자는 약 0.1 초만을 요구한다. 또한 데이터의 크기가 큰 경우, 즉 farm.gml의 경우 Oracle 은 엘리먼트당 평균 검색시간이 약 67초가 요구되나, 본 논문에서 제시한 GML지원 하부 저장 관리자는 약 0.1 초만을 요구한다. 따라서 데이터의 크기에 관계없이 GML지원 하부 저장 관리자는 Oracle 과 비교하여 약 700배 정도 성능이 우수함을 알 수 있다.

마지막으로 (그림 15)는 GML문서의 spatial data 검색시간의 성능을 비교한 것이다. 공간 데이터를 지닌 문서 크기가 작은 road.gml의 경우, Oracle은 공간 데이터 검색시간이 약 0.34초가 요구되나, 본 논문에서 제시한 GML지원 하부 저장 관리자는 약 0.015 초만을 요구한다. 공간 데이터를 지



(그림 15) Spatial Data 검색시간의 성능 비교

닌 문서 크기가 가장 큰 farm.gml의 경우, Oracle은 약 4초가, 본 논문에서 제시한 GML 지원 하부 저장 관리자는 검색을 위해 0.5초가 소요된다. 따라서 spatial data 검색의 경우에는 최소 약 8배에서 최대 약 22배 까지 Oracle 에 비하여 성능이 우수함을 알 수 있다. 결론적으로 모든 평가 부분에서 본 논문에서 제시한 GML지원 하부 저장 관리자가 우수한 것을 알 수 있다. 이러한 이유는 Oracle은 범용적인 응용을 위한 DBMS인 반면, GML 지원 하부 저장 관리자는 GML 지원을 효율적으로 하기 위해 만들어진 GML 문서 전용의 하부 저장관리자이기 때문이다.

6. 결론 및 향후 연구

최근 LBS 및 텔레매틱스(telematics) 응용의 효과적인 지원을 위해, 유클리디언(euclidean)공간 대신, 실제 도로나 철도와 같은 공간 네트워크(spatial network) 를 고려한 연구가 활발하게 수행 중에 있다. 그러나 현재 활발히 진행 중인 공간 네트워크 데이터베이스 연구는 지리 정보의 교환 표준으로 제시된 GML을 지원하는 연구가 매우 부족한 실정이다. 또한 기존의 XML을 저장하기 위한 스키마인 XParent는 공간 지리정보들의 표현의 한계성, 테이블 간의 데이터의 중복, 다수 테이블에 따른 검색의 비효율성의 문제점을 가진다. 따라서 본 논문에서는 XParent의 문제들을 해결하기 위해서 테이블 간의 결합을 고려하여 GML 문서의 효율적인 저장/관리를 위한 GML 저장스키마인 GParent #1, GParent #2, GParent #3를 제안하였다. 제안한 저장스키마의 효율성을 평가하기 위해 GML 문서의 삽입시간, 저장 공간 사용량 및 엘리먼트당 검색시간을 측정하였다. 그 결과 GParent 방법이 삽입시간 및 저장 공간 사용량에서 기존의 XParent 방법보다 우수함을 나타내었고, GML 문서의 삽입시간이 중요한 응용에는 GParent #2 저장 스키마가 최적이고, GML 문서의 검색시간이 중요한 응용에는 GParent #3 저장 스키마가 우수함을 제시하였다. 또한, 제안한 저장스키마에 적합한 GML 지원 하부 저장 관리자를 설계 및 구현하였다. GML 지원 하부 저장 관리자는 GML 문서의 효율적인 검색을 위하여 일반 텍스트 색인 및 공간 지리정보의 색인을 제공하며, 아울러 외부 API를 통해서 사용자가 쉽게 접근할 수 있는 환경을 제공하였다. 하부 저장 관리자의 효율성을 평가

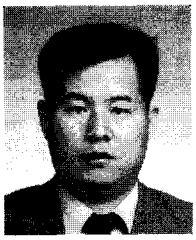
하기 위해, GML 문서의 삽입시간, 엘리먼트당 검색시간 및 Spatial Data에 대한 검색시간을 측정하여 기존 상용 DBMS인 Oracle과 성능비교를 수행하였다. 그 결과 GML 지원 하부 저장 관리자는 모든 평가부분에서 Oracle보다 우수함을 나타내었다. 향후 연구로는 GML 지원 하부 저장 관리자를 in-memory 방식으로 발전시켜서, 보다 효율적인 GML 문서의 저장 및 검색을 통하여 전체적인 성능 향상을 도모하는 것이다.

감사의 글

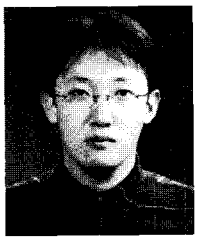
본 과제(결과물)는 교육인적자원부, 산업자원부, 노동부의 출연금으로 수행한 최우수 실험실 지원사업의 연구결과입니다.

참 고 문 헌

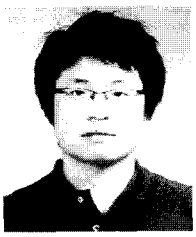
[1] OGC, "Geography Markup Language(GML) Implementation Specification v3.1.1," http://www.opengis.net/gml/, 2004
[2] OGC Specifications, "http://www.opengis.org/techno/specs.html," 1999.
[3] Ron Lake, "http://www.galdosinc.com/technology-whygml.html"
[4] J. Corcoles et al., "Analysis of Different Approaches for Storing GML Documents," Proceedings of the tenth ACM international symposium on Advances in geographic information systems 2002.
[5] F. Tian et al., "The Design and Performance Evaluation of Alternative XML Storage Strategies," SIGMOD record, vol.31, No.1, 2002.
[6] 민준기 외 3명, "다양한 저장소에서의 효율적인 XML 저장기법에 대한 연구," 데이터베이스연구, 제19권 1호 2003.
[7] A. Schmidt et al., "Efficient Relational Storage and Retrieval of XML Documents," In Proceedings of WEBDB 2000.
[8] M. Yoshikawa et al., "Xrel: A path-based approach to storage and retrieval of XML Documents using Relational Databases," ACM Transactions on Internet Technology, Vol.1, No.1, 2001.
[9] Haifeng Jiang et al., "Path Materialization Revisited: An Efficient Storage Model for XML Data," the 2nd Australian Institute of Computer Ethics Conference 2000.
[10] Jayavel Shanmugasundaram, H. Gang, Kristin Tufte, Chun Zhang, David J. DeWitt, and Jeffrey F. Naughton. "Relational databases for querying XML documents: Limitations and opportunities." In VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, Edinburgh, Scotland, pages 302-304, 1999.
[11] P. Bohannon et al., "LegoDB - From XML scheme to relations : a cost-based approach to XML storage," In Proceeding of International Conference on Data Engineering 2002.
[12] Haifeng Jiang et al.. "XParent: An Efficient RDBMS-Based XML Database System.," IEEE 2002
[13] J. Corcoles and P. Gonzalez. A Specification of a Spatial Query Language over GML. ACM-GIS 2001. 9th ACM International Symposium on Advances in GeographicInformation Systems. 2001
[14] H. Jiang, H. Lu, W. Wang and J. Xu Yu. Path Materialization Revisted: An efficient Storage Model for XML Data. 2nd Austrlian Institute of Computer ethics Confrence (AICE2000). Canberra. Australia. 2002.
[15] C. Kanne and G. Moerkotte. Efficient storage of XML data. In proceedings of the international conference on Data engineering. 2000.
[16] A. R. Schmidt, M. L. Kersten, M. A. Windhouwer, and F. Waas. Efficient elational Storage and Retrieval of XML Documents. Workshop on the Web and Databases(WebDB). 2000.
[17] D. Florescu and D. Kossmann. Storing and Querying XML Data Using an RDBMS. Data Engineering Bulletin, 22(3), 1999.
[18] Oracle9i Database Documentation. http://otn.oracle.com:80/docs/products/oracle9i/content.html. 2002.
[19] P. Rigaux, M. Scholl and A. Voisard. Spatial Databases with Application to GIS. Morgan Kaufmann Publishers. 2002.
[20] Berkeley DB "http://www.sleepycat.com"
[21] (주)대경지리정보 "http://www.dkgis.com"
[22] (주)ThinkWare "http://www.thinkwaresys.com"



장 재 우
e-mail : jwchang@chonbuk.ac.kr
1984년 서울대학교 전자계산기공학과 (공학사)
1986년 한국과학기술원 전산학과 (공학석사)
1991년 한국과학기술원 전산학과 (공학박사)
1996년~1997년 Univ. of Minnesota, Visiting Scholar
2003년~2004년 Penn State Univ., Visiting Scholar.
1991년~현 재 전북대학교 컴퓨터공학과 교수
관심분야 : 공간 네트워크 데이터베이스, 상황인식, 하부저장구조



김 영 국
e-mail : uksky@pentaxvoiceware.com
2004년 전북대학교 컴퓨터공학과(공학사)
2004년~2006년 전북대학교 컴퓨터공학과 (공학석사)
2006년~현 재 PENTAX VOICEWARE 연구원
관심분야 : 공간 데이터베이스, 하부저장구조, GIS



김 영 진
e-mail : yzkim@dblab.chonbuk.ac.kr
2007년 전북대학교 컴퓨터공학과(공학사)
2007년~현 재 전북대학교 컴퓨터공학과 석사과정
관심분야 : 공간 데이터베이스, 공간 색인 구조, GIS