효율적인 GML 문서 저장을 위한 저장 스키마의 설계 및 성능평가†

Design and Performance Analysis of Storage Schema for Efficient Storing of GML Documents

장재우* / Jae-Woo Chang, 왕태웅** / Tae-Woong Wang, 이현조*** / Hyun-Jo Lee

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, 그리고 GML 문서의 질의어로 분류된다. GML 문서 저장에 관한 연구는 효율적인 GML 문서 검색을 위해 필수적인 연구이다. 그러나 GML 문서의 저장 스키마에 관한 연구는 거의 전무한 형편이다. 또한 기존 XML 문서 저장 스키마는 공간지리정보 저장에 적합하지 않다. 따라서 본 논문에서는 공간지리정보를 포함한 GML 문서를 효율적으로 저장하기 위한 저장 스키마를 제안한다. 아울러 제안하는 저장 스키마의 성능평가를 실시한다.

Abstract

GML is a mark-up language which is suggested by OGC(Open GIS Consortium) for use of encoding standard concerning with storing and transferring Geographic information. For general spatial network database, researches supporting GML documents can be classified into three categories : parsing, storing, and retrieval of GML documents. Among them, the 'Storing of GML document' is essential for efficient GML document retrieval. However, There is little research on schema for storing GML documents. In addition, the existing schema for storing XML documents can't be used for GML documents due to geographic information. Therefore, In this paper, we propose efficient schema for storing GML documents. In addition, we do performance evaluation of the GML schema.

주요어 : 저장 스키마, GML문서, 공간 네트워크 데이터 베이스
Keyword : Sforage Schema, GML Documents, Spatial Network Databases.

† 이 논문은 교육인적자원부, 산업자원부, 노동부의 출연금으로 수행한 최우수실험실자원사업의 연구결과이며, 아울러 이 연구에 참여한 연구자는 2단계 BK21 사업의 지원비를 받았음.
■ 논문접수 : 2007.4.27 ■ 심사완료 : 2007.6.21
* 전북대학교 컴퓨터공학과 교수(jwchang@chonbuk.ac.kr)
** 삼성전자(주) 재직 중(wkin5g@nate.com)
*** 교신저자 전북대학교 대학원 컴퓨터공학과 석사과정(hjlee@dblab.chonbuk.ac.kr)

1. 서론

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 작용하는 응용 소프트웨어 서비스(OGC), 이동식 사용자가 그들의 위치, 소재 또는 알려진 존재에 대한 서비스를 받도록 하는 것(FCC) 등이 중요한 영역으로 연구되고 있다. 한편 통신과 정보처리를 통합한 기술을 자동차에 제공하는 텔레메틱스(Telemetics) 역시 LBS의 한 영역으로 자리 잡고 있다.

이런 LBS나 텔레메틱스 기술의 핵심은 공통적으로 지리정보를 이용한 서비스를 수행한다. OGC (Open GIS Consortium)는 네트워크, 응용 혹은 플랫폼의 형식에 관계없이 지리정보의 교환 표준으로 GML(Geographic Markup Language)을 채택했다[1]. GML은 피쳐(Feature)라고 불리는 지리적인 실체를 통해서 지리정보를 표현하고 있으며, 공간적인 피쳐와 비공간적인 피쳐를 포함한 지리정보를 전달하거나 저장하기 위해서 XML형태로 코딩한다[2]. GML의 제안자인 Galdos 사는 GML을 지리정보 교환 표준으로 선택할 때 가질 수 있는 다양한 장점을 제시하고 있다[3]. 일반적인 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. 그러나 기존 XML 문서 저장 스키마는 공간지리정보 저장에 적합하지 않기 때문에, 본 논문에서는 공간지리정보를 포함한 GML 문서를 효율적으로 저장하기 위한 저장 스키마를 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 GML과, 기존 연구된 XML과 GML의 저장스키마에 대해서 소개한다. 3장에서는 GML 문서를 효율적으로 저장하기 위한 저장스키마를 제시한다. 4장에서는 본 논문에서 제안한 저장 스키마의 성능평가에 대해서 기술한다. 마지막으로, 5장에서는 결론 및 향후연구를 제시한다.

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 지리 정보의 저장 및 전송을 위한 인코딩 표준으로 GML[1]을 제안하였다. 공간정보와 위치정보를 포함하는 기술들의 상호 운용성(Interoperability)에 목적을 두고 전 세계적으로 수백 개가 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, 2003년 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 피처-특성(Feature-property) 모델을 정의하며, 지오메트리 스키마는 상세한 지오메트리 컴포넌트들을 포함하고, 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(Multi-Geometry) 클래스에 대한 지오메트리 엘리먼트와 좌표를 표현하기 위한 좌표(Coordinate) 엘리먼트

와 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 않기 때문에 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.

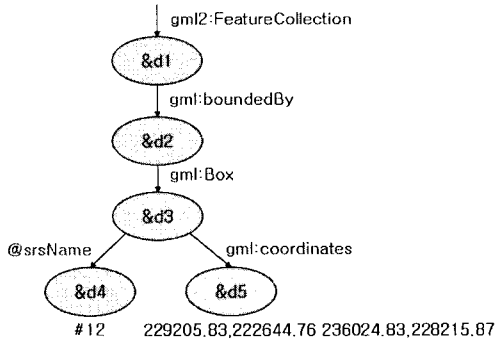
한편 XML/GML문서를 저장하는 또 다른 방법은 XML/GML문서의 구조 정보를 표현하는 DTD 또는 스키마를 기반으로 관계형 데이터베이스의 스키마를 정의하고 정의된 테이블에 XML/GML문서를 저장하는 구조 사상(Structure Mapping)방법이다. 구조 사상 방법은 XML/GML 스키마를 기반으로 생성된 관계형 데이터베이스(Relational database)의 스키마(Schema)에 따라 저장이 이루어지기 때문에, 스키마들을 만족하는 XML/GML 문서들이 많이 존재하거나 스키마 생성에 사용된 스키마의 수정이 빈번하게 발생하지 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 밑줄 친 항목은 Index

```
<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

가 구성된 것을 나타낸다. 즉, LabelPath 테이블은 PathID, Path 항목이 Index로 구성이 되어 있고 Element 테이블은 PathID, DataID 항목이, Data 테이블은 모든 항목인 PathID, DataID, Value 항목이, DataPath 테이블은 Parent Data ID 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 있으며, 각각의 Path 하나에 고유한 PathID를 가지며 루트로부터 단계를 Length로 표현하며 루트 자신은 Length 1을 가진다. 둘째, Element 테이블은 XML 문서에서 모든 Element와 Path 와의 관계를 나타내는 테이블로 각각의 Element는 고유한 DataID를 가지며, 어떤 Path에 부합하는지 표현하기 위해 PathID가 있으며, 같은 Path 일 경우 순위를 나타내는 Order로 구성되어 있다. 셋째, DataPath 테이블은 부모-자식 엘리먼트와의 관계를 나타내는 테이블로 각각의 엘리먼트에 자식 엘리먼트가 존재할 때 Parent Data ID 와 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.

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 5개 테이블

2.4 GParent [13]

기존 GML 문서를 저장하는 연구는 GParent가 있다. GParent는 모델 사상기법 가운데 가장 효율적인 XParent를 확장한 것이며 GML 저장 스키마 GParent#1, 2, 3에 대해서 소개한다.

첫째, XML/GML 문서는 엘리먼트에 데이터가 존재하는 경우가 있다. 하지만 XParent의 경우에는 엘리먼트에 포함된 데이터를 얻어오기 위해 Element 테이블과 Data2 테이블로 부터 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.

```
LabelPath(PathID, Length, Path)
Data-Element(PathID, Ordinal, 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 3> GParent #1 저장스키마

<그림 3>은 GParent #1의 전체 4개의 테이블과 추가된 Index 이다. Data-Element 테이블의 Index 값은 확장된 XParent의 Data2 테이블의 Index 값과 같은 PathID, DataID 항목으로 이루어져 있다. Data-Element 테이블을 사용한 GParent #1의 장점은 자료의 중복현상을 없애고, 질의를 처리할 시에 Element 테이블과 Data 테이블과의 조인 연산 없이 그 결과를 바로 도출할 수 있다. 하지만 이 방법의 단점은 DataPath 테이블을 검색해야만 완전한 엘리먼트를 얻을 수 있다는 점이다.

둘째, XML/GML 문서는 하나의 엘리먼트에 바로 데이터가 있는 경우도 있지만, 그렇지 않고 하위 엘리먼트가 존재할 가능성도 존재한다. XParent의 경우 이를 처리하기 위해서 DataPaht 테이블을 이용해서 하위에 어떤 엘리먼트가 존재하는지 그 DataID를 얻어온다. 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 테이블을 결합하여 하나의 테이블로 표현한다. 이 결합된 테이블을 Element-

```
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 4> GParent #2 저장스키마

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

<그림 5> GParent #3 저장스키마

DataPath 테이블이라 하고, 또한 이렇게 확장된 저장 스키마를 GParent #2 라고 명명한다. <그림 4>는 GParent #2의 전체 4개의 테이블과 추가된 Index 이다. Element-DataPath 테이블의 Index 값은 XParent의 Element 테이블의 Index 값과 같은 PathID, DataID 항목으로 이루어져 있다. Element-DataPath 테이블을 사용한 GParent #2의 장점은 질의를 처리할 시에 Element 테이블에서 하위 엘리먼트가 존재하거나 존재하지 않거나 DataPath 테이블의 검색을 줄일 수 있다는 점이다. 이러한 조인 연산을 줄여 하위 엘리먼트의 DataID를 보다 빠르게 검색할 수 있다. 하지만 이 방법의 단점은 Data 테이블을 검색해야만 완전한 엘리먼트를 얻을 수 있다는 점이다.

아울러 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 불필요한 것은 아니다. 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 5>는 GParent #3의 전체 3개의 테이블과 추가된 Index를 나타낸다. Data-Element-DataPath 테이블의 Index 값은 XParent의 Element 테이블의 Index 값과 같은 PathID, DataID 항목으로 이루어져 있다. Data-Element-DataPath 테이블을 사용한 GParent #3의 장점은 자료의 중복현상을 줄이고 질의 수행 시 하나의 엘리먼트의 검색을 위해서 최대 두 번의 조인 연산을 줄일 수 있다는 점이다. 하지만 이 방법의 단점은 테이블에 삽입되는 레코드의 크기가 증가하면 테이블의 삽입/검색 성능의 저하를 초래한다는 점이다.

마지막으로 관련연구의 장단점을 표로 제시하면 <표 1>과 같고, 이를 통해 기존 관련연구의 문제점을 분석하면 다음과 같다. 첫째, XParent는 데이터의 중복 및 여러 테이블을 참조하는 단점이 존재한다. 둘째, GParent#1과 GParent#2는 XParent의 데이터 중복 문제는 해결하였으나, DataPath, Data 테이블을 각각 검색해야 하는 단점이 존재한

<표 1> 저장스키마의 장단점 비교

저장 스키마	장 점	단 점
XParent	테이블을 의미적으로 구분하여 쉽게 알아볼 수 있다.	데이터의 중복이 많고 검색 시 여러 개의 테이블을 참조해야한다.
GParent1	데이터의 중복과 검색 시 별도의 Data 테이블의 검색을 줄임	DataPath 테이블을 검색해야 한다.
GParent2	데이터의 중복과 검색 시 별도의 DataPath 테이블의 검색을 줄임.	Data 테이블을 검색해야한다.
GParent3	데이터의 중복과 검색 시 별도의 테이블들의 검색을 줄임	하나의 레코드 크기가 증가하므로 삽입/검색 시 성능 저하 현상이 발생할 수 있음.

다. 마지막으로 GParent#3는 GParent#1과 GParent#2의 문제는 발생하지 않으나, 레코드의 크기가 증가하는 단점이 존재한다. 따라서 이러한 단점을 극복할 수 있는 저장 스키마의 설계가 필요하다.

3. GML 문서 저장을 위한 저장 스키마의 설계

3.1 설계 시 고려사항

본 논문에서 GML 문서 저장을 위한 스키마 설계 시 고려사항은 다음과 같다. 첫째, GML문서는 공간 지리정보를 포함하여 다양한 형태로 변환 및 사용이 가능하기 때문에 이미지 도형처리가 가능한 SVG(Scalable Vector Graphics)형태로 변환할 수 있으며 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 브라우징 해야 한다. 따라서 위와 같은 방법으로 사용하기 위해 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 중이며 현재까지 개발된 GML 문서의 스키마 버전은 3.1.1[1]이다. 따라서 기존에 있는 버전 2.0의 GML 스키마로 작성된 문서가 저장되어 있고, 새로운 버전의 GML 스키마로 작성된 문서를 추가하고자 할 경우, 기존 문서나 저장 스키마의 구조적 변경 없이 새로운 문서를 추가하는 것이 필요하다. 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. 아울러 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.

3.2 하위 엘리먼트 정보를 통합한 저장 스키마의 설계

본 절에서는 GML 문서를 위한 새로운 저장 스키마를 설계하기 위해, 기존 모델사상 기법으로 제안된 GParent의 문제점들을 살펴본다. 첫째, GParent 기법 가운데 검색 성능이 우수한 GParent#3는 엘리먼트 검색 시 Join연산을 줄이기 위해 Data Table, Element Table, DataPath Table을 결합하여 Data-Element-DataPath을 사용한다. 그러나 이는 NULL값이 다수 발생하는 단점을 지닌다. 예를 들면 <그림 6>과 같이 DataID &d1은 엘리먼트이며 &d2라는 하위 엘리먼트를 가지고 있을 경우 Type과 Value의 값이 NULL이 된다. 또한 DataID &d5와 같이 Polygon데이터 형으로 DataID의 Type과 Value는 있으나 &d5의 하위 엘리먼트가 없을 경우 ChildID에서 NULL 값을 갖게 된다. 따라서 3개의 테이블을 결합하면 NULL값이 많이 발생하여 저장 공간의 사용량이 비효율적이고 또한 레코드 크기가 증가하므로 삽입/검색 시 성능 저하 현상이 발생할 수 있다.

PathID	Ordinal	DataID	ChildID	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

<그림 6> GParent #3의 Data-Element-DataPath 테이블

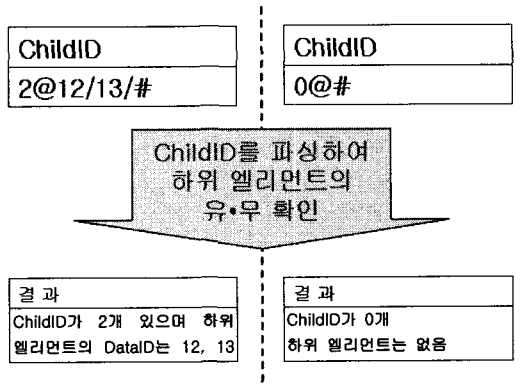
PathID	Ordinal	DataID	ChildID
1	1	&d1	&d2
2	1	&d2	&d3
3	1	&d3	&d4,&d5
4	1	&d4	
5	1	&d5	

<그림 7> GParent#2의 Element-DataPath 테이블

PathID	DataID	Ordinal	Type	ChildID
1	&d1	1	DIR	&d2
2	&d2	1	DIR	&d3
3	&d3	1	DIR	&d4, &d5
4	&d4	1	ATR	NULL
5	&d5	1	Polygon	NULL

<그림 9> 제안하는 저장스키마의 DataPath-Element 테이블

둘째, GParent는 Spatial 데이터의 구분을 위한 'Type'이 정의되어 있으나 하위 엘리먼트가 있는지의 여부는 'Type'에 의해 정의 되어 있지 않아 검색 시 ChildID의 유 · 무를 확인하기 위한 추가적인 연산이 필요하다.



<그림 8> ChildID의 하위 엘리먼트 확인 연산 과정

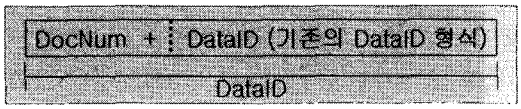
예를 들면 <그림 7>과 같이 하위 엘리먼트가 있는지의 여부가 'Type'으로 정의되어 있지 않기 때문에 <그림 8>과 같은 연산을 해야 하므로 검색에 비효율적이다.

마지막으로, GML 문서를 위한 기존의 저장 스키마는 여러 개의 GML문서를 저장할 경우 기존의 DataID는 엘리먼트를 구분하는 정보만을 표현하고 있으므로, 문서를 구분할 수 있는 ID가 없어 검색 시 각각의 문서에 포함되는 DataID를 구분할 수 있는 방법이 없다.

따라서 본 논문에서는 위와 같은 문제점을 해결하기 위해 GParent를 개선하여 하위 엘리먼트 정보를 포함하고 있는 DataPath, Element 테이블을 통합하여 저장/검색 형태에 알맞은 새로운 저장스키마를 제안한다. 제안하는 저장스키마는 DataPath Table과 Element Table을 결합하여 DataPath-Element Table <그림 9>를 만들어서 첫 번째 문제를 해결하였다. 따라서 NULL값을 최소화하고 데이터의 레코드 수를 줄일 수 있다.

한편 위에서 제시한 두 번째 문제를 해결하기 위한 방법으로 DataPath-Element 테이블에 'Type' 항목을 추가하여 하위 엘리먼트의 유 · 무를 나타내며 Data 테이블에서는 공간(Spatial) 데이터와 비공간(Non-Spatial) 데이터를 구분한다. 아울러 Spatial 데이터 테이블 에서는 공간 데이터의 형식인 점(Point), 선(Line), 면(Polygon)을 표현한다.

마지막으로 위에서 제시한 세 번째 문제인 DataID의 중복과 GML문서를 구분할 수 없는 문제를 해결하기 위해서 문서ID를 추가하는 방법이 있으나, 이 또한 문서ID 데이터의 중복이 많아지는 문제점이 발생한다. 따라서 본 논문에서는 기존의 DataID형식에 GML문서 번호(DocNum)를 추가하여 DataID의 중복성과 문서ID를 구분할 수 없는 문제점을 해결하는 새로운 DataID 형식을 제안한다. 새로운 DataID는 검색 시 원하는 GML문서의 DataID로 검색할 경우 DataID를 Bit & Shift 연산을 수행하여 원하는 문서번호와 DataID를 얻을 수 있는 구조이다. 다음 <그림 10>은 본 논문에서 제안한 새로운 DataID 형식이다.



<그림 10> DataID의 새로운 구조

위와 같은 문제점들을 해결하여 본 논문에서 설계한 GML 저장 스키마 구조는 <그림 11>과 같다. 음영 처리된 부분은 테이블의 색인을 나타내며 이러한 4가지 테이블로 구성된 GML 저장 스키마를 Our-Method라고 명명한다.

DataPath-Element Table

PathID	DataID	Ordinal	Type	ChildID

LabelPath Table

PathID	Level	Path

Data Table

PathID	DataID	Type	Value

SpatialData Table(Rtree)

DataID	Type	Value

<그림 11> 제안하는 저장스키마의 구조

기존 저장 스키마에 있는 Ancestor 테이블은 질의 요청 시 GML문서의 형식으로 변환 할 때 접근하지 않으며 데이터 저장 시 오버헤드와 저장 공간 사용량이 크므로 본 저장 스키마에서는 사용하지 않는다.

따라서 본 논문에서 제안하는 저장스키마의 장점은 DataPath-Element 테이블을 사용하기 때문에, 첫째, NULL값을 줄여 저장 공간을 효율적으로 사용할 수 있고, 둘째, 하위 엘리먼트 정보를 통합하여 질의 요청 시 기존의 스키마보다 GML문서의 형식으로 빠르게 응답할 수 있다. 그러나 단점으로는 Data 테이블을 검색해야하는 오버헤드가 존재한다.

3.3 저장스키마의 예시

본 절에서는 기존 저장 스키마와 본 논문에서 제안한 저장스키마를 실제 GML 문서에 적용시켰을 때의 각 테이블들의 예를 제시한다. 적용할 GML 문서는 <그림 1>의 GML 문서를 사용한다.

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-DataPath테이블

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

PathID	DataID	Ordinal	Type	ChildID
1	&d1	1	DIR	&d2
2	&d2	1	DIR	&d3
3	&d3	1	DIR	&d4, &d5
4	&d4	1	ATR	NULL
5	&d5	1	Polygon	NULL

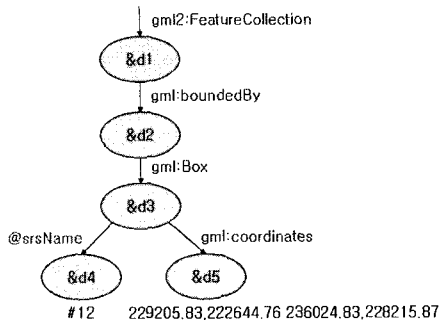
(d) 제안한 저장 스키마의 DataPath-Element 테이블

<그림 12> GParent #1,2,3 과 제안하는 저장스키마 테이블의 예

```
<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 13> GML 문서의 예시 및 문서의 Data Graph

<그림 12>의 (a)는 GParent#1 의 Data-Element 테이블의 예시이다. 그 외의 LabelPath, DataPath, Ancestor 테이블은 확장하지 않았으므로 <그림 2>의 (a),(d),(e)와 동일하며, Non-Spatial Index와 Spatial Index 는 <그림 2>의 (b),(c)와 동일하다. <그림 12>의 (b)는 GParent#2 의 Element-DataPath 테이블의 예시이다. 그 외의 LabelPath, Ancestor 테이블은 확장하지 않았으므로 <그림 2>의 (a),(e)와 동일하며, Data2 테이블, Non-Spatial Index와 Spatial Index 는 <그림 2>의 (a),(b),(c)와 동일하다. <그림 12>의 (c)는 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 Index 는 <그림 2>의 (b),(c)와 동일하다. <그림 12>의 (d)는 기존 저장 스키마의 문제점을 해결하기 위해 DataPath 테이블과 Element 테이블을 결합한 DataPath-Element 테이블이다.

<그림 14>는 본 논문에서 제안한 저장스키마를 실제 GML문서를 적용시켰을 때 각 테이블의 예를 제시한 것이다. 적용할 GML문서는 <그림 13>(a)의 GML문서를 사용한다. <그림 14>(a)는 기존 저장 스키마의 문제점을 해결하기 위해 DataPath 테

PathID	DataID	Ordinal	Type	ChildID
1	&d1	1	DIR	&d2
2	&d2	1	DIR	&d3
3	&d3	1	DIR	&d4, &d5
4	&d4	1	ATR	NULL
5	&d5	1	Polygon	NULL

(a) DataPath-Element 테이블

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

(b) Label Path 테이블

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

(c) Data 테이블

PathID	DataID	Value
5	&d5	229205.83,222644.76 236024.83,228215.87

(d) Spatial Data 테이블

<그림 14> 제안하는 저장 스키마의 4개 테이블

이블과 Element 테이블을 결합한 DataPath-Element 테이블이며, 나머지 테이블은 모든 데이터 값이 들어있는 Data 테이블, 공간지리정보의 색인을 위해 Spatial Data 테이블, LabelPath 테이블로서 기존 GParent 저장스키마 형식과 동일하다.

4. 성능평가

본 절에서는 본 논문에서 제안한 저장스키마의 효율성을 알아보기 위해 기존 저장스키마 XParent, GParent #1,2,3과 성능평가를 실시한다. 효율성에 대한 항목으로는 GML 문서의 삽입 시간, 저장 공간 사용량, GML 문서 형식으로 반환하기 위한 하위 엘리먼트 전체 검색, 부분 검색을 위한 특정 하위 엘리먼트 검색과 특정 상위 엘리먼트 검색 및 Spatial 데이터에 대한 검색 시간을 측정한다. 성능평가 DBMS는 디스크 기반으로 구성된 GigaBase[14]를 사용하였다. 성능평가 환경은 Intel Pentium4 2.4GHz CPU (메모리 Ram 1GB)에서 동작하는 Windows Server 2003 운영체제 하에서, 컴파일러로 Visual C++ 7.1을 사용하여 수행하였다. 입력 데이터는 두 종류의 데이터를 사용한다. 첫째, GML Document Type A는 (주)대경지리정보[15]에서 제공한 전라북도 남원시 덕과면 고정리의 데이터이고, GML Document Type B는 (주)Thinkware[16]에서 제공한 서울시(5개구)와 전라북도 및 전주, 익산, 군산의 데이터이다. 또한 Type A 문서의 특징은 문서의 단계가 깊으며, Non-Spatial Data 보다 Spatial Data를 보다 많이 포함하고 있는 반면 Type B 문서의 특징은 문서의 단계가 깊지 않으며, Spatial Data 보다 Non-Spatial Data를 다소 많이 포함하고 있다. <표 2>는 본 실험에서 사용한 데이터를 나타낸다.

4.1 GML 삽입 성능평가

본 절에서는 기존의 GParent 저장 스키마와 본 논문에서 제시한 저장스키마(OurMethod라 명명

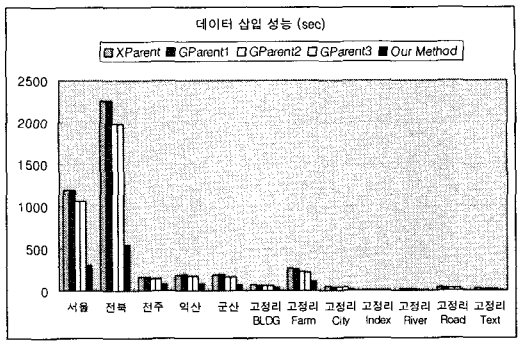
<표 2> 실험에서 사용한 데이터

문서구분	문서 이름	문서 크기	엘리먼트수 (공간자료포함수)	비 고
Type A	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
	chonbuk.gml	103MB	9,250,890 (260,321)	전라북도 데이터
	seoul.gml	46.4MB	540,282 (155,473)	서울시 데이터
Type B	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

한다)의 삽입 성능 평가를 시행한다. 삽입한 데이터는 <표 2>의 열한가지 데이터를 가지고 성능평가를 수행하였다.

<표 3>은 Type A, B의 GML문서들의 삽입시간을 나타낸 것이며 <그림 15>는 GML문서의 삽입시간 성능을 비교한 것으로 Type A, B GML문서들의 삽입시간을 나타낸 그래프이다.

<그림 15>에서 보듯이 데이터가 Type A인 BLDG일 경우 삽입시간은 Our-Method는 약 38sec, GParent#3는 약 61sec, GParent#2는 약 62sec, GParent#1은 약 65sec, XParent는 약 67sec와 같이 측정된다. 또한 데이터가 Type B인 전주일 경우 삽입시간은 Our-Method는 약 80sec, GParent#3는 약 141sec, GParent#2는 약 138sec, GParent#1은 약 161sec, XParent는 약 163sec와 같이 측정된다. 삽입 성능이 좋은 GParent#2에 비해 본 논문에서 제안한 저장 스키마가 약 1.5배의 삽입 시간 성능이 우수하다. 이는 제안한 저장 스키마가 DataPath-Element 테이블을 사용 NULL값을 줄여 저장 공간을 효율적으로 사용하기 때문이다. 또한 제안한 저장 스키마는 기존 저장 스키마에 있는 Ancestor 테이블을 사용하지 않기 때문에 데이터 삽입 시 오버헤드가 작으므로 삽입성능이 우수한 것을 알 수 있다.



<그림 15> GML 문서 삽입시간 비교

4.2 저장 공간 사용량

본 절에서는 기존의 GML 저장 스키마와 본 논문에서 제안한 저장 스키마의 저장 공간 사용량 성능 평가를 수행한다.

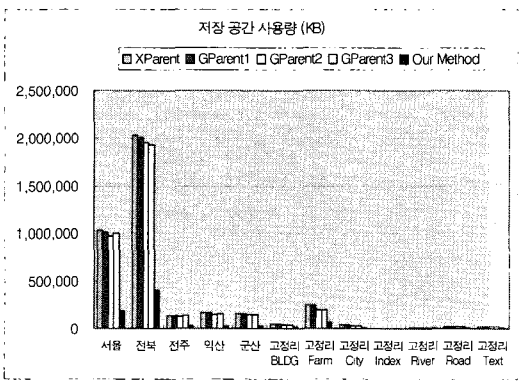
<표 4>는 Type A, B GML문서들의 저장 공간 사용량을 나타낸 것이며 <그림 16>은 GML문서의 저장 공간 사용량 비교한 것으로 Type A, B GML

<표 3> GML 문서 삽입시간

GML Doc Type	Data	XParent	GParent 1	GParent 2	GParent 3	Our Method
Type A	서 울	1191.69	1195.27	1063.53	1064.70	304.30
	전 북	2252.41	2254.72	1978.15	1980.01	538.92
	전 주	163.70	161.06	145.67	141.59	80.38
	익 산	188.06	189.15	172.40	176.45	83.15
	군 산	187.45	194.75	166.75	164.98	70.85
Type B	고정리 BLDG	67.01	65.01	62.12	56.14	38.99
	고정리 Farm	264.64	254.34	222.00	210.13	112.39
	고정리 City	51.30	41.81	39.81	41.89	22.75
	고정리 Index	12.70	12.01	10.15	9.25	7.85
	고정리 River	22.33	18.40	17.75	15.12	9.63
	고정리 Road	35.73	32.90	26.58	29.87	13.16
	고정리 Text	32.33	23.75	21.30	20.66	11.47

<표 4> GML 문서 저장 공간 사용량(KB)

GML Doc Type	Data	XParent	GParent 1	GParent 2	GParent 3	Our Method
Type A	서 울	1,029,312	1,013,568	974,904	998,512	187,520
	전 북	2,032,144	2,014,464	1,950,440	1,933,632	400,672
	전 주	140,528	141,080	128,576	136,688	26,632
	익 산	167,536	170,344	153,096	158,768	32,008
	군 산	161,856	164,624	147,888	153,368	30,256
Type B	고정리 BLDG	48,608	48,616	43,272	43,344	15,228
	고정리 Farm	256,336	250,608	204,896	199,568	70,848
	고정리 City	39,256	38,488	35,112	34,600	13,224
	고정리 Index	4,312	4,304	4,228	4,228	4,208
	고정리 River	13,128	12,912	12,048	11,904	7,152
	고정리 Road	24,912	24,360	22,376	21,936	10,616
	고정리 Text	17,696	17,168	15,928	15,552	7,576



<그림 16> GML 문서 저장 공간 사용량 비교

문서의 저장 공간 사용량을 나타낸 그래프이다.

<그림 16>에서 보듯이 익산 데이터의 Our-Method는 32,008KB, GParent#3는 158,768KB, GParent#2는 153,096KB, GParent#1은 170,344KB, XParent는 167,536KB와 같이 측정된다. Type A, B 모두 기존 GML 저장 스키마보다 본 논문에서 제안한 저장 스키마의 저장 공간 사용량이 우수함을 알 수 있다. 이는 DataID를 찾기 위한 GParent#3의 익산 Data-Element-DataPath테이블 용량은 7,900KB이며 OurMethod의 DataPath-Element테이블 용량은 4,109KB이다. 따라서 DataPath-Element 테이블을 사용하여 데이터의 NULL값을 줄여 저장 공간을 효율적으로 사용하기 때문이다. 또한 제안한 저장 스키마는 기존 저장 스키마에 있는 Ancestor 테이블을 사용하지 않기 때문에 데이터 저장 시 저장 공간 사용량이 우수함을 알 수 있다.

4.3 하위 엘리먼트 전체 검색

본 절에서는 기존 저장스키마인 XParent, GParent와 본 논문에서 제안한 저장스키마의 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 검색하여서 PathID를 얻는다. 그 후 XParent의 경우에는 Element 테이블을, GParent #1의 경우에는 Data-Element 테이블을, GParent #2의 경우에는 Element-DataPath 테이블을, GParent #3의 경우에는 Data-Element-

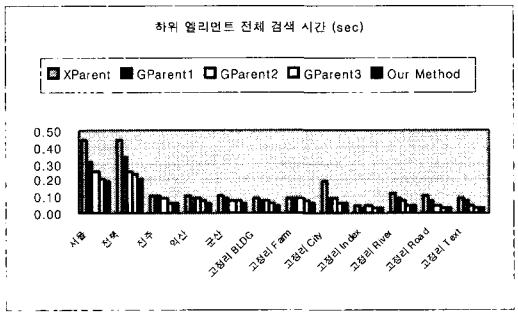
<표 5> GML 문서의 하위 엘리먼트 전체 평균 검색 시간(sec)

GML Doc Type	Data	XParent	GParent1	GParent2	GParent3	Our Method
Type A	서 울	0.448	0.308	0.247	0.205	0.187
	전 북	0.441	0.334	0.254	0.231	0.203
	전 주	0.109	0.101	0.093	0.062	0.058
	익 산	0.106	0.086	0.081	0.067	0.062
	군 산	0.109	0.082	0.076	0.071	0.062
Type B	고정리 BLDG	0.093	0.071	0.078	0.062	0.046
	고정리 Farm	0.093	0.087	0.081	0.071	0.064
	고정리 City	0.193	0.082	0.093	0.062	0.062
	고정리 Index	0.046	0.034	0.046	0.031	0.031
	고정리 River	0.125	0.091	0.075	0.046	0.046
	고정리 Road	0.109	0.078	0.046	0.031	0.031
	고정리 Text	0.093	0.078	0.046	0.031	0.031

DataPath 테이블을, 마지막으로 Our-Method 테이블의 경우에는 DataPath-Element 테이블을 검색하여서 DataID를 얻는다. 이렇게 얻어진 DataID를 통해 하위 엘리먼트가 존재하는지, 또는 값을 가지고 있는지의 검색을 통해 GML 문서에 해당하는 하위 엘리먼트들을 결과로 얻는다.

본 논문에서 비교하고자 하는 5개의 저장스키마의 가장 큰 차이점은 Data 테이블, Element 테이블, Data Path 테이블의 조합으로 이루어진 Data-Element, Element-DataPath, Data-Element-DataPath와 DataPath-Element 테이블 이므로 DataID를 통한 하위 엘리먼트 전체 검색을 평가 대상으로 삼는다. 이를 위한 평가 방법은 전주 데이터에서 DataID를 검색하여 DataID의 하위 엘리먼트 전체를 검색 GML 문서로 반환하는 평균 검색 시간을 측정하는 것이다.

<표 5>는 Type A, B의 GML문서 DataID에 대한 하위 엘리먼트 전체의 평균 검색시간을 나타낸 것이며 <그림 17>은 GML문서들의 하위 엘리먼트 전체의 평균 검색시간을 비교한 것으로 Type A, B의 하위 엘리먼트 전체를 검색하여 GML 문서로 반환하는 평균 검색시간을 나타낸 그래프이다.



<그림 17> 하위 엘리먼트 전체 평균 검색 시간

데이터가 전주의 경우 Our-Method는 0.058sec, GParent#3는 0.062sec, GParent#2는 0.093sec, GParent#1은 0.101sec, XParent는 0.109sec와 같이 측정된다. 문서의 Type에 상관없이 기존의 저장 스키마 에서는 GParent #3의 경우가 좋은 성능을 보이고 있으며 본 논문에서 제안하는 저장 스키마의 성능이 가장 우수함을 알 수 있다. 이는 제안하는 저장스키마의 DataID와 하위 엘리먼트를 찾기 위해 2개의 테이블을 합한 DataPath-ElementTable의 크기가 GParent#3에 3개의 테이블을 합한 Data-Element-DataPath보다 작아

<표 6> 특정 하위 엘리먼트 검색 시간(sec)

GML Doc Type	Data	XParent	GParent1	GParent2	GParent3	Our Method
Type A	서 울	0.289	0.227	0.193	0.175	0.152
	전 북	0.367	0.279	0.221	0.211	0.184
	전 주	0.062	0.046	0.041	0.041	0.036
	익 산	0.067	0.051	0.046	0.036	0.031
	군 산	0.067	0.046	0.041	0.031	0.026
Type B	고정리 BLDG	0.051	0.046	0.041	0.036	0.036
	고정리 Farm	0.062	0.046	0.036	0.031	0.031
	고정리 City	0.031	0.026	0.026	0.020	0.026
	고정리 Index	0.015	0.010	0.010	0.010	0.010
	고정리 River	0.041	0.036	0.031	0.020	0.026
	고정리 Road	0.067	0.041	0.036	0.031	0.031
	고정리 Text	0.041	0.041	0.041	0.031	0.031

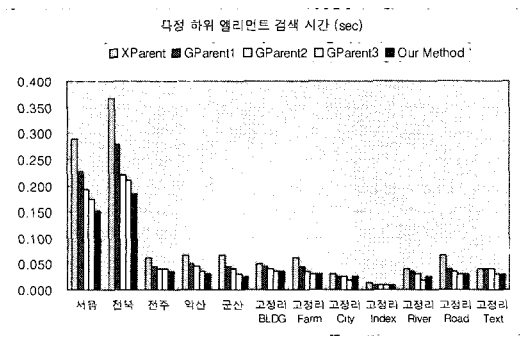
DataID의 하위 엘리먼트를 검색하는 질의 응답시간이 낮기 때문이다. 또한 DataPath-ElementTable에 'Type' 항목을 추가하여 DataID의 'Type'을 알기 위해 ChildID를 구분하는 연산을 하지 않기 때문이다. 따라서 하위 엘리먼트 전체 평균 검색시간이 우수한 것을 알 수 있다.

4.4 특정 하위 엘리먼트 검색

본 절에서는 GML 문서의 특정 하위 엘리먼트 검색을 수행한다. 이를 위한 평가 방법은 DataID를 질의하여 질의한 DataID의 특정 하위 3번째 엘리먼트까지 검색한 응답시간을 측정하였다.

<표 6>은 Type A, B의 GML문서 DataID의 하위 3번째 엘리먼트까지의 평균 검색시간을 나타낸 것이며, <그림 18>은 GML문서들의 특정 하위 엘리먼트까지의 평균 검색시간을 비교한 것으로 Type A, B의 특정 하위 엘리먼트를 찾는 평균 검색시간을 나타낸 그래프이다.

<그림 18>의 데이터 Type B의 경우 GML문서의 크기가 작아 GParent#3와 유사한 응답 성능을 보인다. 반면 Type A와 같이 GML 문서의 크기가 클 경우 익산 데이터의 Our-Method는 0.031sec, GParent#3는 0.036sec, GParent#2는 0.046sec, GParent#1은 0.051sec, XParent는 0.067sec와 같이 측정되었다. 이는 DataID를 찾기 위해 사용하는 제안하는 저장스키마의 DataPath-Element 테이블이 GParent#3의 3개 테이블을 합한 Data- Element-DataPath테이블보다 크기가 작고 'Type'을 추가하여 하위 엘리먼트를 연산하는 시간을 줄여 질의 응답시간이 줄어든다. 따라서 제안하는 저장스키마가 특정 하위 엘리먼트 검색에서 성능이 우수함을 알 수 있다.



<그림 18> 특정 하위 엘리먼트 평균 검색 시간

<표 7> 특정 상위 엘리먼트 검색 시간(sec)

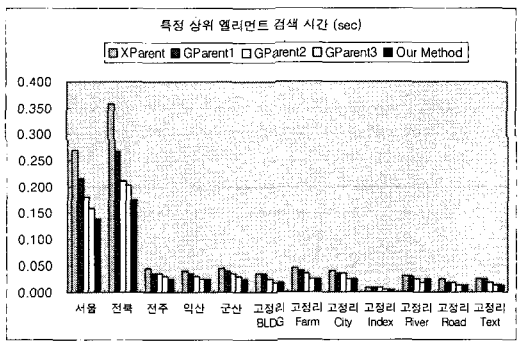
GML Doc Type	Data	XParent	GParent1	GParent2	GParent3	Our Method
Type A	서 울	0.269	0.216	0.179	0.158	0.139
	전 북	0.358	0.268	0.211	0.202	0.175
	전 주	0.046	0.036	0.036	0.031	0.026
	익 산	0.041	0.036	0.031	0.026	0.026
	군 산	0.046	0.041	0.036	0.031	0.026
Type B	고정리 BLDG	0.036	0.036	0.026	0.020	0.020
	고정리 Farm	0.046	0.041	0.036	0.026	0.026
	고정리 City	0.041	0.036	0.036	0.026	0.026
	고정리 Index	0.010	0.010	0.010	0.005	0.005
	고정리 River	0.031	0.031	0.026	0.020	0.026
	고정리 Road	0.026	0.020	0.020	0.015	0.015
	고정리 Text	0.026	0.026	0.020	0.015	0.015

4.5 특정 상위 엘리먼트 검색

본 절에서는 GML 문서의 부분 검색을 위한 특정 상위 엘리먼트 검색을 수행한다. 이를 위한 평가 방법은 DataID를 질의하여 질의한 DataID의 상위 3번째 엘리먼트까지 검색한 응답 시간을 측정하였다.

<표 7>은 Type A, B의 GML문서 DataID의 특정 상위 엘리먼트까지의 평균 검색시간을 나타낸 것이며 <그림 19>는 GML문서들의 특정 상위 엘리먼트를 찾는 평균 검색시간을 비교한 것으로 Type A, B의 특정 상위 엘리먼트를 찾는 평균 검색시간을 나타낸 그래프이다.

<그림 19>의 데이터 Type B의 경우 GML문서의 크기가 작아 GParent#3와 유사한 응답 성능을 보인다. 반면 Type A와 같이 GML 문서의 크기가 클 경우 서울 데이터의 Our-Method는 0.139sec, GParent#3는 0.158sec, GParent#2는 0.179sec, GParent#1은 0.216sec, XParent는 0.269sec와 같이 측정되었다. 이는 DataID를 찾기 위해 사용하는 제안하는 저장스키마의 DataPath-Element 테이블이 GParent#3의 3개 테이블을 합한 Data- Element-DataPath테이블보다 크기가 작아 질의 응답시간이 줄어든다. 따라서 제안하는 저장스키마가 특정 상위 엘리먼트 검색에서 성능이 우수함을 알 수 있다.



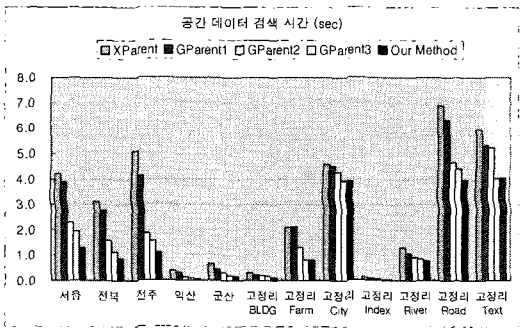
<그림 19> 특정 상위 엘리먼트 평균 검색 시간

4.6 Spatial 데이터 검색

본 절에서는 Spatial 데이터 검색시간의 성능을 평가를 수행한다. 이를 위한 검색 방법은 모든 Spatial Data를 찾을 수 있는 충분한 크기의 영역 질의를 통해서 DataID의 좌표 값을 얻는 시간을 측정하였다.

<표 8> Spatial 데이터 평균 검색 시간(sec)

GML Doc Type	Data	XParent	GParent1	GParent2	GParent3	Our Method
Type A	서 울	4.218	3.893	2.297	1.929	1.312
	전 북	3.092	2.783	1.595	1.127	0.874
	전 주	5.093	4.125	1.859	1.562	1.109
	익 산	0.406	0.328	0.14	0.093	0.062
	군 산	0.64	0.468	0.312	0.187	0.171
Type B	고정리 BLDG	0.312	0.244	0.203	0.156	0.142
	고정리 Farm	2.093	2.109	1.296	0.828	0.819
	고정리 City	4.562	4.495	4.293	3.912	3.889
	고정리 Index	0.125	0.093	0.071	0.031	0.031
	고정리 River	1.312	1.093	0.921	0.853	0.796
	고정리 Road	6.906	6.343	4.652	4.421	3.953
	고정리 Text	5.953	5.328	5.218	4.031	4.046



<그림 20> Spatial 데이터 평균 검색 시간

<표 8>은 Type A, B의 GML문서에 Spatial 데이터 평균 검색시간을 나타낸 것이며 <그림 20>은 GML문서의 Spatial 데이터 검색시간 성능을 비교한 그래프이다.

<그림 20>에서 보듯이 전주데이터의 경우 Our-Method 1.109sec, GParent#3 1.562sec, GParent#2 1.859sec, GParent#1 4.125sec, XParent 5.093sec이며 고정리 Road의 경우 Our-Method 3.953sec, GParent#3 4.421sec, GParent#2 4.652sec, GParent#1 6.343sec, XParent 6.906sec이다. 이는 Spatial데이터를 검색한 결과는 영역 내에 있는 DataID와 하나의 좌표를 가져오기에 문서의 'Type'과 상관없이 특정 엘리먼트 검색 성능이 좋은 본 논문에서 제안하는 저장스키마가 우수함을 알 수 있다.

본 논문에서 제시한 저장스키마는 전체적으로 그 성능이 우수함을 알 수 있다. 제시한 저장스키마에서는 DataPath-Element 테이블을 사용하여, NULL값을 줄였으며, 또한 하위 엘리먼트 정보를 통합하였다. 이러한 장점은 Data의 크기가 클 경우 GPrarent#3에 비해 저장 공간 사용량을 약 20% 정도까지 줄일 수 있어, Data 테이블을 검색해야하는 단점에도 불구하고, 검색 시간을 줄일 수 있었다. 그러나 고정리 City, 고정리 River처럼 Data의 크기가 작을 경우에는 저장 공간이 GPrarent#3에 비해 약 40% 정도로 줄어들어, 이로 인해 Data 테이블을 검색해야 하는 단점을 상쇄하지 못하고, 특정 엘리먼트 검색시 GParent#3 보다 좋지 않은 검색 성능을 보였다. 이와 같이 본 논문에서 제시한 저장스키마는 Data의 크기가 작을 경우 특정 엘리먼트 검색시 GParent#3 보다 좋지 않은 성능을 보일 수 있는 제한점을 가지고 있다.

5. 결론 및 향후 연구

최근 LBS 및 텔레매틱스(telematics) 응용의 효과적인 지원을 위해, 이상적인 유클리디언(Euclidean)공간 대신, 실제 도로나 철도와 같은 공간 네트워크(spatial network) 를 고려한 연구가 활발하게 수행 중에 있다. 그러나 현재 활발하게 진행 중인 공간 네트워크 데이터베이스 연구는 지리정보의 교환 표준으로 제시된 GML을 지원하는 연구가 거의 없는 실정이다.

따라서 본 논문에서는 GML 문서의 효율적인 저장/관리를 위해 기존의 GML 저장스키마인 GParent를 개선하여 새로운 저장스키마를 제안하였다. 또한, 제안한 저장스키마의 효율성과 범용성을 알아보기 위해 기존의 저장스키마와 성능평가를 수행하여 문서의 단계가 깊고 용량이 클수록 저장 공간, 삽입 성능, 검색의 성능이 좋음을 보였다. 향후 연구로는 본 논문에서 제안한 저장 스키마를 실제 응용 프로그램에 적용하여, 제안하는 저장스키마의 실용성을 검증하는 것이다.

참고문헌

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.", Proceedings of the 18th International Conference on data Engineering 2002
13. 김영국, "GML문서 저장을 위한 저장 스키마 및 하부 저장 시스템의 설계 및 구현", 전북대학교 컴퓨터 공학과 석사 학위 논문, 2006.
14. GigaBase "http://www.garret.ru/%7Eknizhnik/gigabase.html"
15. (주)대경지리정보 "http://www.dkgis.com"
16. (주)ThinkWare "http://www.thinkwaresys.com"

장재우
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~현재 전북대학교 컴퓨터공학과 교수
관심분야 : 공간 네트워크 데이터베이스, 상황인식, 하부저장구조

왕태웅
2005년 우석대학교 전산통계학과(학사)
2007년 전북대학교 대학원 컴퓨터공학과(공학석사)
2007년~현재 삼성전자(주) 재직 중
관심분야 : 공간 데이터베이스, 질의처리 알고리즘, 공간 색인 구조

이현조
2006년 전북대학교 컴퓨터공학과(공학사)
2006년~현재 전북대학교 대학원 컴퓨터공학과 석사과정
관심분야 : 데이터 마이닝, 공간 데이터베이스, 공간 색인 구조