碩士學位論文

닷넷 프레임워크 기반 XML문서의 검색 성능 평가

濟州大學敎 大學院

電算統計學科

姜 淳 喆

2004年 1月

닷넷 프레임워클 기반 XML문서의 검색 성능 평가

지도교수 박 경 린

강 순 철

이 논문을 이학 석사학위 논문으로 제출함.

2004년 1월

강순철의 이학 석사학위 논문을 인준함.

심사위원장 _____ (인)

위 원 _____ (인)

위 원 _____ (인)

제주대학교 대학원

2004년 1월

A Retrieval Performance evaluation about the XML Documents based on .Net Framework

Soon Chol Kang

(Supervised by professor Kyung-Leen Park)

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF SCIENCE.

DEPARTMENT OF COMPUTER SCIENCE AND STATISTIC
GRADUATE SCHOOL
CHEJU NATIONAL UNIVERSITY

DECEMBER 2003

목 차

그 림 차 례

표 차 례

Abstract

The thesis evaluates the performance of retrieval on XML Documents based on .Net Framework. The techniques used for the retrieval are DOM, SAX, XPath, and ADO.NET. DOM was proposed by W3C while SAX was originated by XML-Dev mailing list members. XPath along with XSLT was also proposed by W3C. ADO.NET was used when database is involved

When using DOM, the retrieval speed for the XML document consisting of the *element* only is about 20%~30% faster than that for the XML document consisting of the *attribute* only. When using SAX, on the other hand, the retrieval speed for the XML document consisting of *attribute* only is about 30%~40% faster than that for the XML document consisting of *element* only. When using XPath, the retrieval speed for the XML document consisting of *attribute* only is two times faster than that for the XML document consisting of *element* only. When using ADO.NET, the retrival using DataReader is about 10% faster than that using DataSet.

In the evaluation of the retrieval speed on the XML document based on .NET Framework, the conclusion can be summarized as followings. First, in the case DOM technique, we will see good retrieval performance if the XML document is constructed mainly by the *element*. Second, in the case of SAX and XPath technique, we will see good retrieval performance if the XML document is constructed mainly by the *attribute*. At last, the SAX technique shows better retrieval performance than other techniques in general.

Ⅰ 서 론

XML(eXtensible Markup Language)은 W3C(World Wide Web Consortium) 산하 XML Working Group에서 인터넷 표준으로 1996년에 제안된 언어로서 간단하고, SGML(Standard Generalized Markup Language)에서 파생된 매우 유동적인 문서 포맷이다. HTML의 한계를 극복하고 SGML의 다양한 기능성은 수용하고 문법 및 구조는 단순화시킨 XML은 처음에는 거대한 크기의 전자 출판을 위해 디자인되었었지만, 웹과 다른 여러 분야에서 폭 넓고 다양한 데이터를 교환하는 중요한 역할을 하게 되었으며, 이 중요성은 점점 커지고 있다.

XML은 HTML에서 사용되는 태그의 일종인 엘리먼트(ELEMENT)와 어트리뷰트(ATTRIBUTE), 엔티티(ENTITY)라는 기본적인 표현 방식을 사용하는데, 이것은 HTML 표현에서와는 다르게 사용자가 임의적으로 문서 구조를 구성 및 변경을 할 수 있게 되는 즉, 자기-서술적(Self-describing)인 문서 구조 표현이 가능하게 되어 자유롭게 문서의 내용을 표현할 수 있으며, 반복적인 사용으로 구조화된 문서로 확장할 수도 있다.

구조화된 문서인 XML문서의 검색 및 변경을 위한 접근 및 조작에 대한 방법이 필요하게 되는데, W3C에서는 문서에 대한 객체 정의를 표현하는 DOM(Document Object Model) Level 1 명세서를 1998년 10월에 발표하였다. DOM은 HTML과 XML 문서를 위한 응용 프로그래밍 인터페이스(API)이며, DOM은 문서를 접근하고 조작하기 위한 방법으로 문서의 논리적 구조를 정의한다. DOM은 구조적 문서를 트리 기반의 계층적 구조로 재구성할 수 있도록 했는데 구조적인 정적 문서를 트리 기반의 동적 구조로 재구성하기 때문에 문서를 파싱한 후에 메모리에 트리가 만들어지므로 인터페이스를 통해 쉽게 문서의 내용 및 구조를 수정할 수 있다.

DOM의 대안으로 XML문서에 대한 접근을 용이하게 하기 위해 David Megginson을 중심으로 XML-Dev라는 메일링 리스트의 구성원들이 SAX(Simple API for XML)라는 기술을 개발하였다. SAX는 파서로부터의 이벤트를 캐치(Catch)할 수 있도록 선언된 특

정 메소드를 필요로 하는 이벤트 구동(Event Driven) 인터페이스로써, 부하없이 문서를 검색할 수 있게 되는 장점이 있지만 각각의 이벤트에 대해서 사용자가 처리해 줘야 하는 부담이 발생한다.

보편적으로 응용 프로그램을 개발하는 사용자들은 데이터를 보관하기 위해 데이터베이스 시스템을 사용하고 있으며, 이런 데이터베이스 시스템들은 상당한 양의 데이터를 저장 및 보관, 검색을 할 수 있는 기능 등을 제공하고 있을 뿐만 아니라, 백업과 복구라는 기능까지도 기본적으로 제공해 주고 있다.

하지만, 기존 데이터베이스 시스템들은 관계형 데이터베이스 시스템으로써 정규화된 형태로만 데이터가 저장되기 때문에 중첩적이고 비정규적인 객체 중심으로 표현되어진 XML문서를 저장하기에 적합하지 않은 형태를 갖게 된다. 이런 불편함을 해결해 보고자 XML문서를 관계형 데이터베이스에 저장하기 위한 방법에 대한 많은 연구들이 이루어지고 있으며, 또한 객체관계형 데이터베이스 시스템들이 개발되어 있기도 하여 XML문서에 특별한 변경 및 변환 혹은 그런 기능을 수행하는 미들웨어 등을 거치지 않고서 바로 시스템에 저장도 할 수 있게 되었다.(이기훈, 2003)

본 논문에서는 차세대 웹 문서 표준 언어인 XML, XML문서의 객체화 기술인 DOM과 이벤트 드리븐 방식을 사용하는 SAX를 이용하여 XML문서에서의 특정 데이터를 검색하는 성능 평가를 실시하였으며, 또한 관계형 데이터베이스 시스템에 XML문서에 포함되어 있는 데이터를 저장하여 검색 속도를 측정하고 XML문서에서 행하여진 성능 평가와 비교 분석해 보도록 하겠다. 성능 평가를 위해 C#이라는 프로그램 언어를 이용하여 응용 프로그램을 작성하고, DOM과 SAX 그리고, 데이터베이스에 대한 검색 속도 측정을 위해 닷넷 프레임워크의 클래스들을 이용하게 된다.

본 논문의 Ⅱ절에서는 XML과 XML관련 API인 DOM과 SAX관련 기술에 대한 개요 및 현황 분석, DOM과 SAX의 비교, XPath 질의어 기술과 XML문서 저장 기법 및 저장 시스템, 닷넷 프레임워크에 대해 소개할 것이고, Ⅲ절에서는 Testbed 구축 방법 및 위의 기술들을 이용한 XML문서의 검색 성능 평가 및 관계형 데이터베이스 시스템과의 성능 평가 및 비교 분석에 대한 내용을 기술한다. Ⅳ절에서는 각 조건에 대한 실험 결과를 보이고, Ⅵ절에서는 도출된 결론과 추후의 연구 과제를 제시한다.

Ⅱ. 관련 연구

1. XML

XML(eXtensible Markup Language)은 웹 상에서 구조화된 문서를 전송 가능하도록 설계된 표준화된 마크업 언어이다. XML은 인터넷에서 기존에 사용되어 오던 HTML (HyperText Markup Language)의 한계를 극복하고, SGML(Standard Generalized Markup Language)의 복잡함을 단순화함으로써, SGML과 HTML 양쪽 모두와의 상호 운용성 및 용이한 구현 환경을 제공한다(정희경, 1999)

XML이 처음 등장하였을 때만 하여도 XML이 기존에 웹에서 사용하는 HTML을 대신할 것인가 또는 HTML과 공존하며 사용될 것인가에 대한 매우 초보적인 논의가 있었으나 현재는 XML을 기반으로 하는 프로그래밍 기술이 발전함에 따라 인터넷, 전자상거래, 음악, 과학, 디지털 도서관 등과 같은 매우 다양한 분야에서 새로운 응용에 XML을 적용하고 있으며, 이러한 추세로 XML은 당당히 차세대 웹상의 표준 문서 포맷으로 자리잡아가고 있다(이강찬, 2001)

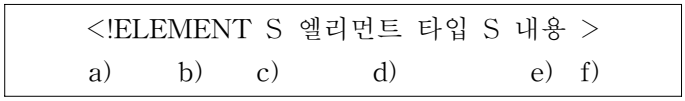
1) XML 문서 구조(정희경, 1999)

XML의 문서구조는 논리적 구조와 물리적 구조로 구성된다. 논리적 구조는 문서의 전체적인 구조를 표현하는 부분으로 엘리먼트, 속성 등의 구성요소를 이용하여 표현하며, 물리적 구조는 엔티티를 이용하여 표현한다. 이들이 혼합되어 문서의 구조를 나타내는 DTD(Data Type Definition)를 형성하는데, 다음에 DTD를 구성하는 각 요소들에 대해 설명한다.

(1) 엘리먼트 선언 (Element Declaration)

엘리먼트 선언은 엘리먼트 형과 내용으로 정의되는데 이는 XML문서의 전체적

인 구조를 나타내는 부분이다. 엘리먼트 선언을 위한 형식은 (그림 1)과 같다.



(그림 1) 엘리먼트 선언 형식

a) '<!' : 마크업 선언 개방 구분자

b) 'ELEMENT' : 엘리먼트 선언을 위한 예약어

c) 'S' : 구분자 (separator)

d) '엘리먼트 타입' : 엘리먼트 공통 식별자(GI)

e) '내용 : 엘리먼트 내용

f) '>' : 마크업 선언 폐쇄 구분자

엘리먼트 선언에 따른 엘리먼트 선언 예를 (그림2)에 보인다.

```
<!ELEMENT MEMO   (TO, FROM, BODY) >
<!ELEMENT TO     (#PCDATA)        >
<!ELEMENT FROM   (#PCDATA)        >
<!ELEMENT BODY   (P)*             >
<!ELEMENT P      (#PCDATA | Q)*   >
<!ELEMENT Q      (#PCDATA)        >
```


(그림 2) 엘리먼트 선언 예

(그림 2)에서 엘리먼트 선언 예의 내용에서 MEMO엘리먼트는 하위 엘리먼트로 서브 엘리먼트 TO, FROM, BODY가 순서적으로 나올 수 있고 P엘리먼트는 내용이 #PCDATA, Q중의 구성요소 중에서 하나도 나오지 않을 수도 있고, 반복해서 나올 수도 있다.

(2) 속성 선언(Attribute Declaration)

속성 선언은 문서나 엘리먼트의 속성을 정의하는 것으로 엘리먼트 형과 속성의 이름, 데이터형, 디폴트 값으로 지정한다. 속성 선언 형식은 (그림 3)과 같다.

```
<!ATTLIST  S  엘리먼트명  S?  속성정의목록 >
    a)          b)          c)
```

(그림 3) 어트리뷰트 선언 형식

a) 'ATTLIST' : 속성 선언을 위한 예약어

b) '엘리먼트명' : 선언할 속성을 가질 엘리먼트명

c) '속성정의목록' : 속성이름, 형, 디폴트값으로 구성

속성형은 문자열형(string type), 토큰형(tokenized type), 열거형(enumeration type)의 3가지로 크게 나누어진다. 문자열형은 CDATA의 텍스트 유형이고, 토큰형의 속성값은 참조값의 목록을 선언하는 IDREF나 IDREFS, 속성값을 현재 선언된 부분이나 데이터 엔티티명, 엔티티명들의 목록이 될 수 있도록 선언하는 ENTITY, ENTITIES, 속성값을 명칭 토큰이나 명칭 토큰들의 목록으로 선언하는 NMTOKEN이나 NMTOKENS의 형이 있다. 열거형은 표기법 속성값들이나 명칭 토큰들을 괄호 안에 선언하는 값들의 목록들로 구성한다. 속성 선언에서 선언되는 각 속성의 디폴트 값에는

- 명확히 인식 할 수 있는 값
- REQUIRED : 속성값이 반드시 입력되어야 함.
- IMPLIED : 속성값이 정의되지 않았다면 응용 프로그램에서 그 값을 부여하는 경우
- FIXED : 고정된 디폴트 속성값만을 갖도록 하는 경우

(3) 엔티티 선언(Entity Declaration)

SGML에서와 마찬가지로 XML에서도 엔티티는 문서 내에서 참조될 수 있는 문자 집합의 단위로 일반 엔티티(general entity)와 매개변수 엔티티(parameter entity)로 크게 나눌 수 있다.

엔티티는 (그림 4)와 같이 선언한다.

```
// 일반 엔티티
    <!ENTITY  S  명칭  S  엔티티정의  S? >
       a)            b)            c)
// 매개변수 엔티티
    <!ENTITY S  %명칭  S  엔티티정의  S? >
```

(그림 4) 엔티티 선언 형식

a) 'ENTITY' : 엔티티 선언을 위한 예약어

b) '명칭' : 엔티티 명

c) '엔티티 정의' : 대체될 엔티티 정의

일반 엔티티의 경우 XML문서 작성에서 자주 사용되는 문자열이 있다고 할 때, 예로 'Cheju National University'문자열이 문서에서 많이 사용될 때 매번 이 문자열을 쓰는 것이 아니고 이를 일반 엔티티로 선언하여 참조에 사용하면 편리하다. 이때 먼저 엔티티를 다음과 같이 선언하는데 이는 CNU가 'Cheju National University'와 같음을 뜻한다. XML문서 중에 "Cheju National University"라는 문자열이 들어가야 할 부분에 &CNU;를 입력하면 "Cheju National University"로 확장된다.

```
<!ENTITY  CNU  "Cheju Nation University" >
```

(그림 5) 일반 엔티티 선언 예

매개변수 엔티티 선언 예는 다음과 같다.

```
<!ENTITY  %TS  TITLE, SUBJECT >
<!ELEMENT BEFORE(%TS;, NAME, DATE, E-MAIL? ) >
```

(그림 6) 매개변수 엔티티 선언 예

(4) 표기법 선언(Notation Declaration)

표기법은 외부 이진 엔티티 형식 명칭에 의해 식별되는데 이것은 비XML 데이터와 외부 응용 프로그램을 연결해 주는 역할을 한다.

표기법 선언 예는 다음과 같다.

```
<!NOTATION gif SYSTEM
"C:\Program Files\Windows NT\Accessories\ImageVue\kodakimg.exe">
```

(그림 7) 표기법 선언 예

2) XML 문서(정희경, 1999)

XML문서는 XML 버전을 지정하는 선언으로 시작하며, 첫 번째 시작태그 이전에 선언되어야 한다. XML내의 마크업 기능은 DTD에 의해 제공되는데 이에 따라 문서를 작성하면 '유효한 XML문서'가 된다. 실제 XML 문서의 생성은 DTD를 갖지 않고 XML문법에 따라 구성된 문서(Well-Formed Document)와 DTD에 따라 작성된 문서(Valid Document)로 생성한다.

(1) Well-Formed 문서

Well-Formed 문서는 DTD가 존재하지 않은 인스턴스라도 XML 구문에 맞게 태그된 문서를 말한다. 그 예는 (그림 8)과 같다.

```
// Well-Formed Document
<?xml version=1.0?>
<conversation>
<greeting> Hello, World! </greeting>
<response> Hello, XML! </response>
</conversation>

// Not Well-Formed Document
<?xml version=1.0?>
<convesation>
<greeting> Hello, World! </greeting>
<response> Hello, XML! <response>
```

(그림 8) Well-Formed 문서 예

(2) Valid 문서

Well-Formed 문서와는 달리 Valid 문서는 DTD를 가져야 할 뿐만 아니라, XML 규칙 및 정의된 DTD규칙에 따라야 한다. 이것은 XML 문서가 전형적으로 생성되고 갱신되는 양식이다. 그 예는 (그림 9)와 같다.

```
// DTD와 인스턴스를 별개 파일로 분리
<?xml version=1.0 encoding=UTF-8?>
<!DOCTYPE sample SYSTEM sample.dtd>
<greeting> Hello, World! </greeting>

// DTD와 인스턴스를 한 파일로 관리
<?xml version=1.0?>
<!DOCTYPE sample [
<!ELEMENT greeting (#PCDATA) >
]>
<greeting> Hello, World! </greeting>
```

(그림 9) DTD의 예

2. API

XML 문서를 다루는 API(Application Programming Interface)에는 두 가지가 존재한다. DOM(Document Obeject Model)과 SAX(Simple API for XML)가 그것인데, DOM은 문서를 접근하고 조작하기 위한 방법으로 구조적 문서를 트리 기반의 계층적인 논리적 구조를 정의하는 반면에 SAX(Simple API for XML)는 문서의 스트림 내의 태그를 탐지해서 문서를 다루는 이벤트 드리븐 방식을 사용한다.

1) DOM

DOM은 XML문서를 위한 API로써, W3C의 Working Group에서 문서에 대한 객체 정의를 표현하는 DOM(Document Object Model) Level 1 명세서를 1998년 10월에 발표하였다. 또한, CSS(Cascading Style Sheet) 모델과 이벤트 모델 및 질의(Query)등에 대한 인터페이스를 추가한 DOM Level 2 명세서를 1999년 3월에 발표하였으며 DOM Level 3에 대한 명세서가 2003년 8월에 발표되었다.

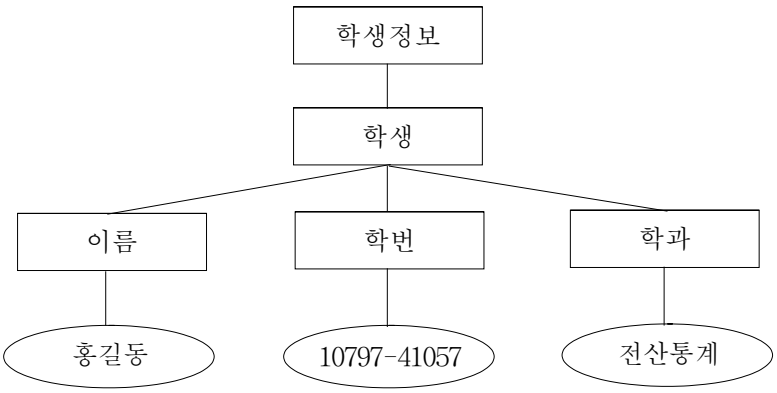
DOM은 XML 문서를 접근하고 조작하기 위한 방법으로 문서의 논리적 구조를 정의하게 되는데 사용자들은 문서를 생성하고 그 문서의 구조에 따라 항해(Navigator)하고, 엘리먼트와 어트리뷰트 등을 추가, 변경, 삭제 할 수 있다(이강찬. 2001)

즉, DTD 또는 마크업 정보 등을 포함하는 구조적 문서를 대상으로 그 구성요소들을 객체화하여 문서의 구조와 내용에 대한 접근 및 조작을 지원하는 언어 독립적인 트리 기반의 인터페이스를 제공한다(선승상. 2000)

DOM은 XML 문서 전체를 파싱한 후에 트리 구조로 문서 트리를 구성한다. (그림10)에서 XML문서의 예를 보이고, (그림 11)에서는 (그림10)에서 보여지는 XML문서 내에 정의된 엘리먼트들을 각각 하나의 객체로 모델링한 후에 DOM 트리 구조로 구조화한 모습이다.

```
<?xml version="1.0" encoding="EUC-KR"?>
<학생정보>
    <학생>
      <이름>홍길동</이름>
      <학번>10799-41057</학번>
      <학과>전산통계</학과>
    </학생>
</학생정보>
```


(그림 10) XML 문서 예



(그림 11) DOM 트리

DOM은 노드 인터페이스, 서브클래스 엘리먼트, 속성, 문자데이터 인터페이스를 노드로부터 상속 받아 정의한다. 노드 인터페이스는 가장 기본적인 타입으로써 노드의 컴포넌트에 접근하기 위한 메소드를 정의하는데, 부모노드와 정해지지 않은 순서적 자식 노드를 가질 수 있다.

또한, 새로운 노드를 만들기 위해 노드 값을 변경하기 위한 메소드를 제공한다. 따라서, 어플리케이션은 이 인터페이스를 이용하여 문서를 재구성 할 수 있으므로, DOM은 XML에 대한 데이터 뷰라고 할 수 있다. 즉 XML 문서의 특정 엘리먼트의 자식 엘리먼트의 텍스트 내용을 알고 싶다면, DOM 인터페이스를 이용하면 된다. DOM은 시스템 리소스를 많이 차지하므로 XML 문서의 파일크기가 크면 문제가 될 수 있지만 노드의 생성, 삭제 등에 대한 풍부한 API를 제공하여 보다 쉽게 XML문서에 대해 접근하게 되어 원하는 작업을 할 수 있도록 해 주고 있다.

DOM이 XML문서를 읽어서 트리 구조로 주기억장치에 갖고 있으며, 이때 각각의 노드는 엘리먼트, 텍스트가 되며, DOM을 기반으로 하는 XML 프로세서는 DOM을 이용하여 구문 트리를 만들고 전체 문서를 다루기 위한 어플리케이션 프로그램을 다룬다(김노환, 2001)

DOM은 문서의 구조를 표현하는 인터페이스들의 원시 데이터인 Node 인터페이스를 상속함으로써, 문서 내의 엘리먼트들을 노드화 및 객체화하고, 이에 대한 조작 인터페이스를 제공한다. DOM은 XML 문서를 계층적 구조의 노드 집합으로 구성하는데, Document, Element, Attribute, Text, Processing Instruction, CDATASection 및 Comment 등의 XML 문서의 구성 요소들은 원시 객체인 Node를 상속받는데 실체는 원시 데이터형 별로 다르지만 동일한 노드로 표현된다. XML 문서는 단일 Document 노드를 가지는 논리적 구조로 구성되는데, Document 노드는 부모 노드를 갖지 않는 최상위 루트 노드로서 여러 개의 자식 Element 노드들을 가질 수 있다(선승상, 2000)

(표 1)는 DOM의 인터페이스에 대한 간략한 설명이다.

인터페이스 이름	설명
Document	XML 문서 전체를 의미하는 인터페이스
DocumentType	DTD를 지원할 수 있도록 DTD의 entity와 notation에 접근하는 인터페이스
Node	XML 문서의 모든 노드를 정의하는 인터페이스
NodeList	노드의 집합을 의미하는 인터페이스
NamedNodeMap	노드의 이름을 통해 노드를 다룰 수 있도록 하는 인터페이스
Element	XML문서의 엘리먼트 노드를 정의하는 인터페이스
Attr	XML문서의 어트리뷰트 노드를 정의하는 인터페이스
Text	컨텐츠에 들어가는 텍스트를 정의하는 인터페이스
CDATASection	CDATASection을 정의하는 인터페이스
Entity	엔터티를 정의하는 인터페이스
EntityReference	DTD에 선언된 Entity를 참조하도록 하는 인터페이스
Notation	Notation을 정의하는 인터페이스

(표 1) DOM인터페이스의 예

2) SAX

SAX는 David Megginson을 비롯한 XML개발자들이 처음 만들었으며, 그 이후 많은 연구가 이루어지면서 XML 문서를 처리하기 위해 만들어졌지만 표준기구인 W3C에서 XML문서를 다루는 기술의 표준으로 구분하지는 않았다. SAX1.0은 1998년 5월에 처음 구현되어져 발표되었고, 2000년 5월에 SAX1.0을 보강한 SAX2.0이 발표되어 현재까지 사용되고 있다.

SAX는 SAX파서로부터의 이벤트를 감지할 수 있도록 선언된 특정 메소드를 필요로 하는 이벤트 드리븐 방식의 인터페이스로써, DOM과 마찬가지로 XML문서를 접근하여 수정하거나 삭제, 삽입하는 기능을 한다. SAX는 문서의 시작부터 끝까지를 스트림으로 불러오는데 DOM과는 다르게 메모리에 트리를 만들지 않으면서 문서를 파싱하게 되므로, 부하가 없는 문서를 파싱하는 방법을 가지게 된다. 또한, 모든 문서를 읽기 전에 이

벤트를 캐치하는 메소드 중 어느 하나가 어플리케이션에 통제를 반환할 수 있으므로, 전체 문서를 처리하는 일을 피할 수 있게 되고, 실행 시에 파서가 건내주는 항목들을 무시하거나 변경을 할 수 있게 되어 문서의 필요한 부분만 처리할 수 있게 된다.

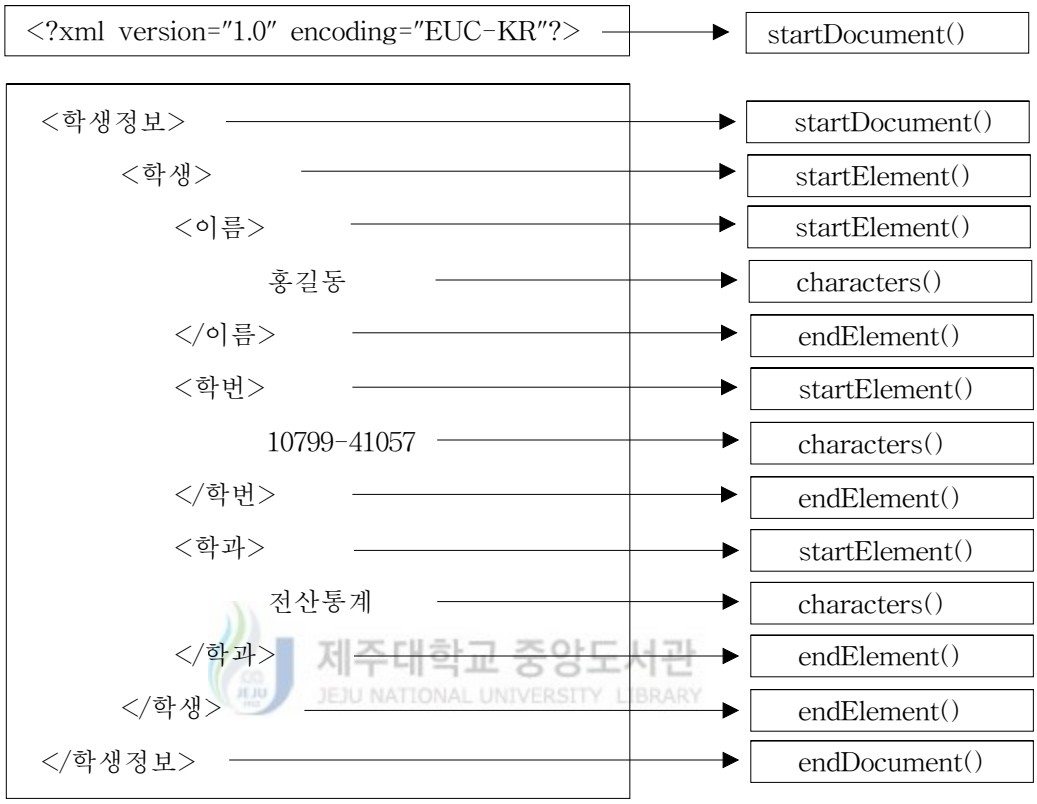
(표 2)에서는 SAX의 인터페이스들에 대한 간략한 설명이다.

인터페이스	설명
XMLReader	콜백(Callback)을 사용하여 XML문서를 읽기 위한 인터페이스
ContentHandler	문서와 엘리먼트의 시작과 끝, 컨텐츠 텍스트나 PI에 대한 파서 이벤트를 담당하는 인터페이스
DTDHandler	DTD에 포함된 표기법과 외부 엔티티에 대한 정보를 담당하는 인터페이스
Entity Resolver	Entity Resolver는 공개 ID(URN)를 시스템 ID(URL)로 변환하도록 허용하는 함수를 정의하는 인터페이스
ErrorHandler	SAX 응용프로그램 처리 과정의 에러 핸들러를 위한 인터페이스

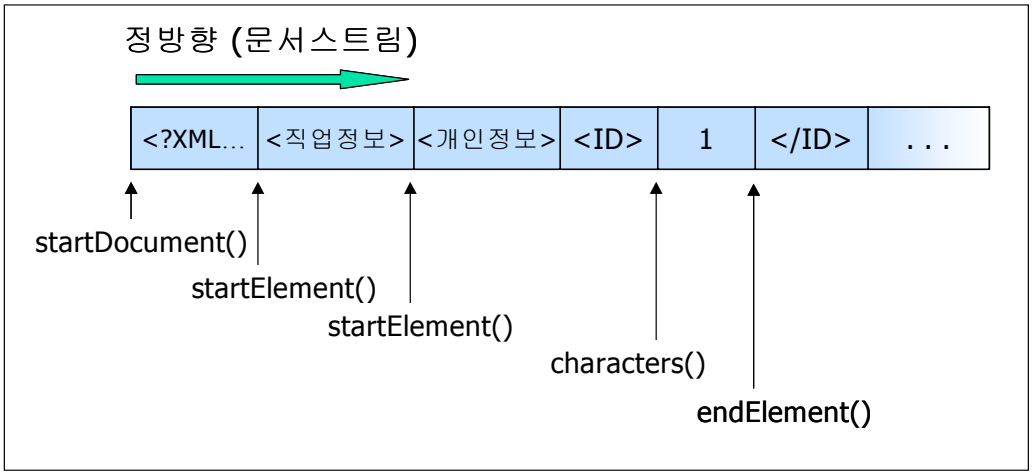
(표 2) SAX 인터페이스의 예

SAX파서의 동작 원리는 XML문서를 읽어들일 때 발생하는 이벤트를 감지하여 동작하도록 되어 있는데, 문서 혹은 엘리먼트의 시작을 만나면 startElement이벤트를 발생시키고, 끝을 만나면 endElement이벤트를 발생시켜 ContentHandler에 의해 인식한 후 응용 프로그램에서 적절하게 프로그래밍하는 방식으로 XML문서를 처리하게 된다.

(그림 12)는 닷넷에서 사용되는 SAX 파서의 동작원리에 대한 설명이다.



(그림 12) SAX 파서의 동작 원리



(그림 12) SAX의 동작 원리

3) DOM과 SAX의 비교

DOM과 SAX는 XML 문서를 어플리케이션에서 다룰 수 있는 API를 제공하는 공통점이 있는데, 이를 정리해 보면 다음과 같다.

먼저, DOM은 XML문서를 메모리에 적재함과 동시에 파서에 의해 트리 구조로 구성되고 이 문서 트리에 접근하여 추가 및 삭제, 변경을 하게 되지만 SAX는 XML문서를 스트림으로 규정하여 메모리에 적재시키지 않으면서 스트림을 앞에서부터 뒤까지 읽어가게 되고 이때 문자열들을 판단하여 이벤트를 발생시키는 이벤트 드리븐 방식을 사용한다. 따라서, DOM에서는 XML문서를 메모리에 적재시켜 접근하기 때문에 사용자 입장에서는 쉽게 접근하여 핸들링하게 할 수 있는 장점을 지니게 되지만 문서의 크기가 큰 경우에는 부하가 많이 걸리는 약점이 있게 되고, 반면에 SAX는 문서를 읽으면서 바로 처리하기 때문에 DOM에 비해 처리 속도가 빠르게 되며, 메모리를 사용하지 않기 때문에 부하가 적게 걸리는 장점을 지니게 된다. 하지만, SAX는 메모리에 존재하는 트리를 탐색할 수는 없으며, 스트림 방식으로 문서를 불러오게 되므로 문서의 앞과 뒤를 옮겨다니는 데에는 적합하지 않게 되는 단점도 가지게 된다.

3. XPath

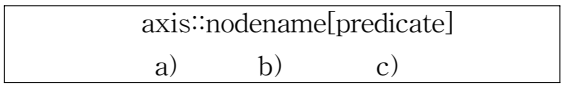
XPath는 1999년 11월 W3C에서 XSLT와 함께 XPath 1.0 권고된 트리 탐색 언어이다. 이 것은 XML문서의 일부분을 지시하기 위해 사용되는데 URL 경로 표기법을 사용하여 XML문서의 계층적인 구조를 논리적으로 탐색하는데 이 구조는 요소 노드(element node), 속성 노드(attribute node), 값 노드(text node)를 사용하여 트리 구조로 형성된다. 또한 XPath는 각 노드의 문자열 값을 계산하는 방법을 정의하고 있다(J.Clark, 1999)

XPath는 노드의 경로를 나타내기 위해 위치 경로(Location Path)를 사용하는데, 경로 위치 표현 방식은 절대경로와 상대 경로의 두 가지 방식이 있다. 절대경로는 문서의 루트 노드 즉 '/'로부터 시작하여 경로를 지시하는 방식이고, 상대경로는 현재의 노드 위치를 기준으로 해서 다른 노드의 상대적인 노드의 경로를 나타내는 방식이다. 이 때 경

로 위치를 기초부(basis)와 술어부(predicate)로 나누어 설명할 수 있는데, 기초부는 축(Axis)과 노드 검사 부분으로 구성된다. 여기서 축이란 문맥 노드(Context Node)와 경로 단계로 선택된 노드간의 관계를 명시한다. 한편, 술어부는 대괄호([,])로 묶인 표현식을 말하는데, 축에 관련된 노드 집합을 새로운 노드 집합으로 만들기 위한 필터링 작업을 한다.

현재까지 XPath 질의에 대한 연구와 이를 이용한 시스템들에 대한 많은 연구가 이루어져 오고 있다(김노환, 2001)(고영기, 2002)(김경원, 2002)

XPath에서의 Location path의 형식은 (그림 15)와 같다.



(그림 13) Location Path형식

a) 검색하고자 하는 키가 되는 Location path이름

b) 검색하고자 하는 실제 노드 이름

c) 조건식을 나타내며, Xpath에서 제공되는 연산자와 함수를 이용하여 구성

(표 3)은 anxi에 올 수 있는 Location Path 표현식이고, (그림 17)은 단축 표현식이다.

anxi name	설 명
ancestor	현재 노드의 모드 부모 노드들을 가져온다.
ancestor-or-self	모든 부모노드(조상)와 현재 노드를 포함하여 가져온다.
attribute	현재 노드의 모든 속성을 가져온다.
child	현재 노드의 모든 자식노드를 가져온다.
descendant	현재 노드의 모든 자식노드을 가져온다.
descendant-or-self	모든 자식노드와 현재의 노드를 포함하여 가져온다.
following-sibling	현재 노드 다음에 있는 모든 형제노드를 가져온다.
namespace	현재 노드의 모든 네임스페이스 노드를 가져온다.
parent	현재 노드의 부모노드를 가져온다.
preceding-sibling	현재 노드 전에 있는 모든 형제노드를 가져온다.
self	현재 노드를 가져온다.

(표 3) Location Path 표현식

단축표현식	의미	예
@	attribute::	“개인정보[@직업=“통계학자”]” : 개인정보의 속성 중에 직업이 통계학자인 것
.	self::node()	“.//개인정보 : 현재 노드인 개인정보 엘리먼트
..	parent::node()	“../개인정보” : 개인정보 엘리먼트의 부모 엘리먼트
//	/descendant-or-self::node()	“//직업” : 현재 위치에서 자식 노드가 개인 정보인 모든 자식 노드
/	Document root	“/개인정보” : 문서의 Root엘리먼트의 하위에 있는 개인정보 엘리먼트

(표 4) Location Path 단축 표현식

4. XML 문서의 저장

XML 문서는 기본이 되는 엘리먼트를 기준으로 하여 트리 구조로 구성되어 있으며, 이런 구조는 현재까지 많이 쓰이고 있는 관계형 데이터 베이스 시스템에 저장하기에는 적합지 않은 구조이다. 이런 점을 해결하기 위해 데이터베이스 기술 분야에서는 방대한 양의 데이터를 웹을 통해 공유하기 위해서 XML문서를 효율적으로 DBMS에 저장하고 검색하는 기술에 대한 연구가 활발히 진행되고 있다.(D. Florescu 등, 1999)(J. Shanmugasundaram, 1999)(김훈 등, 2000)

1) 문서 저장 기법에 따른 연구

XML문서는 기본적으로 하나의 문서일 뿐이다. 이 문서의 내용들이 데이터로써 쓰여지기 위해서는 저장시스템에 저장이 되어야 하며 검색이 되어야 한다. 이런 저장 및 검색을 위한 제안된 방식들을 아래에 기술한다.

(1) 리스트 형태를 이용한 방법

각 엘리먼트 노드들은 자신의 고유 번호를 갖고 있고 트리의 상위 노드는 하위 노드들의 고유번호를 리스트 형태로 갖고 있다. 이렇게 자신의 하위 노드에 대한 번호를 이용하여 트리 구조에서의 검색이나 서브 트리에 대한 구조적 검색을 할 수 있다. 트리 구조에서 특정 노드를 찾기 위해서는 트리 항해를 하여 탐색해야 하므로, 많은 탐색 시간이 소요되며, 특정 노드의 하위 서브 트리에 대한 질의나 트리 구조에서의 특정 레벨에 관련된 질의를 하기 어렵거나 효율이 낮아 문서의 운용 데이터베이스로 사용하기에는 부적합하다.

(2) 경로 엘리먼트 ID(Path Element ID)를 이용한 방법

경로 ID는 자신이 자식 엘리먼트 중 몇 번째 엘리먼트인가를 표시하는 엘리먼트 ID를 자신의 조상에 대한 엘리먼트 ID로부터 계속 이어받아 구성된다(이용석 등, 1998) 즉, 자기 자신에 대한 접근 경로가 된다. 이와 같은 경로 엘리먼트 ID를 사용할 경우, XML 문서의 트리 구조상 어느 위치의 엘리먼트라도 쉽게 검색해 낼 수 있다. 그러나 반복되는 엘리먼트가 존재하면 경로 엘리먼트 ID가 무한히 증가되는 경우가 발생하므로 관계형 데이터베이스에 저장하는 것이 어렵다(이용석, 1998)

(3) DFS(Depth First Search) Numbering을 이용한 방법

트리의 루트 노드로부터 DFS 방식으로 노드를 방문하여 처음 방문할 때의 순서와 자신의 자식 노드의 방문이 모두 끝난 뒤의 노드의 방문 순서를 1쌍으로 구성한다(이용석 등, 1998) 이렇게 구성된 DFS Numbering 순서쌍은 문서의 트리 구조상에서 특정 노드에 속해있는 하위 트리에 대한 질의를 하나의 SQL 문장으로 처리할 수 있다. 하지만 몇 단계 위 또는 몇 단계 아래의 특정위치 엘리먼트에 대한 질의 처리는 DFS Numbering 하나만으로는 어렵고 부가적인 속성 필드를 필요로 하거나, 추가적인 검색 및 연산이 필요하다. 또한 문서를 데이터베이스에 추가나 삭제 시 문서를 트리로 구성하여 DFS Numbering 한 뒤에 저장해야 하는 작업이 필요하다.

(4) 분할 저장 기법

분할 저장 기법은 XML 문서의 내용을 엘리먼트 단위로 나누어서 저장하고 검색 시 구조 정보를 참조하여 해당 요소 노드나 하위 엘리먼트들을 재구성하여 검색 결과를 반환하는 기법을 말하는데 사용하는 테이블의 개수에 따라 단일 테이블에 모든 정보를 저장하는 단일 에지 테이블 기법과 여러 개의 테이블 집합을 이용하여 저장하는 관계형

테이블 집합 방식이 있다(D. Florescu, 1999)(J. Shanmugasundaram, 1999)

(5) 가상 분할 저장 기법

가상 분할 기법은 XML 문서 전체 내용을 하나의 BLOB(Binary Large Object)형으로 데이터베이스에 저장하고 XML 문서의 엘리먼트들을 추출하고, 그 엘리먼트가 실제 문서상에 위치 정보를 표현하기 위해 시작 오프셋과 종료 오프셋을 사용하여 표현한다.

가상 분할 기법은 검색 효율이 우수하지만 문서 일부분이 갱신되었을 때 오버헤드가 크기 때문에 일반적으로 검색 위주의 응용 개발 시스템에 주로 사용된다(D. Florescu, 1999)(J. Shanmugasundaram, 1999)

2) 하부 저장 시스템에 따른 연구

XML문서를 저장하기 위한 시스템으로 기존에 연구되어 왔던 데이터베이스 시스템이 많이 쓰이고 있고, 위에서 기술한 방식으로 저장이 되어 검색이 되어 지고 있다. 이런 XML 문서를 저장하기 위한 시스템에 대한 기존의 연구를 기술한다.

(1) 파일 시스템

파일 시스템은 XML 데이터를 저장하기 위한 가장 단순한 방법으로 엘리먼트를 구성하는 서브 엘리먼트들 간의 그룹화가 가능하다는 장점을 가지지만 XML 데이터에 대한 질의가 어렵다는 단점이 있다.

(2) 관계형 데이터베이스 시스템

이 것은 XML 문서의 반구조적인 형태를 관계형 데이터 베이스 시스템의 테이블 형태로 변환하여 저장하는 방식을 사용하는데, 이 방법은 기존의 관계형 데이터 베이스의 우수한 성능을 유지하면서 활용할 수 있다는 장점이 있지만, 여전히 XML문서의 반구조적인 트리 구조를 자연스럽게 테이블 형태로 표현하기에는 힘들다는 점과 질의 처리시에 조인 연산이 고비용을 초래하는 단점이 있다.

(3) 객체 지향 데이터베이스 시스템

OOP(Object Oriented Programming) 개념을 이용한 것으로 XML 문서의 트리 구조에서 각각의 노드를 객체 개념으로 이해해서 표현한 것으로서, 관계형 데이터베이스 시스템에 비해 모델링 측면에서 좀더 쉽게 모델링을 할 수 있게 되어 자연스러운 장점이 있지만, 현재의 연구(Christophides, V 등, 1994)에서는 XML 문서 구조와 동일한 구조의 스키마만을 생성하므로 입력으로 쓰이는 XML 문서의 구조가 저장과 검색에 효율적인 형태가 아니라면 그에 따른 저장과 검색성능이 떨어지는 단점이 있다.

(4) XML 전용 데이터베이스 시스템

XML이 반구조적인 데이터(Semistructured Data)의 범주에 속하기 때문에 이런 데이터를 위한 독자적인 시스템을 확장하여 XML 문서에 포함되어 있는 데이터를 저장하고자 하는 접근 방식으로서 대표적인 시스템으로는 Stanford 대학에서 개발한 Lore가 있다(J. McHugh 등 1999). 이 방식은 XML이 가지고 있는 반구조적인 특성을 잘 표현하고 활용할 수 있다는 장점을 지닌다. 그러나 이러한 접근 방법의 문제점은 지난 20년간 연구되어 온 여러 가지 데이터 베이스 시스템의 기능을 활용하는데 어려움이 있다(J. Shanmugasundaram, 1999)

(5) 객체관계형 데이터베이스 시스템

기존의 시스템들에 비교해서 볼 때 XML문서구조의 트리 구조를 좀더 자연스럽게 표현할 수 있다는 점과 대용량 데이터에 대한 복잡한 질의를 안정적으로 처리할 수 있게 해 주는 점, 현재까지의 데이터베이스 관련 연구 기술들을 활용할 수 있는 장점이 있다. 현재 이 시스템과 관련한 연구들이 활발하게 진행되고 있다.(정태선, 2002)(이기훈, 2003)

5. 닷넷 소개과 닷넷 프레임워크

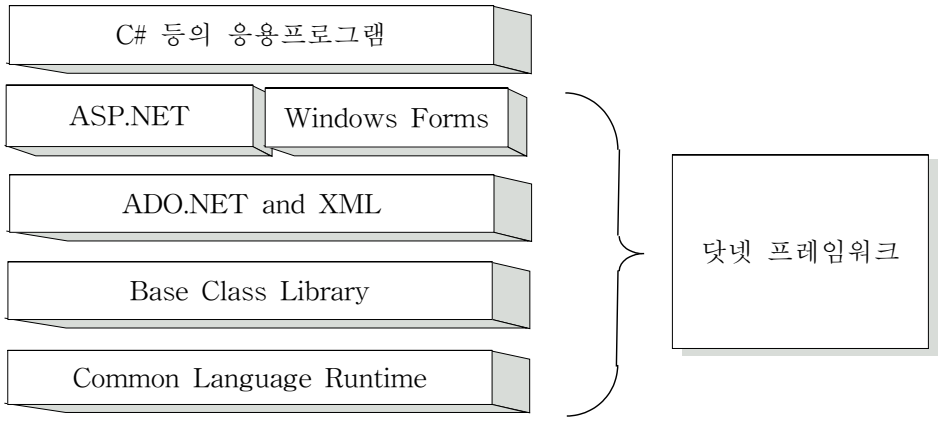
1) 닷넷의 소개

닷넷은 닷넷 프레임워크와 Visual Studio.NET(이하 비쥬얼 닷넷)을 포함한 마이크로 소프트 사(이하 MS사)의 사업적 전략 등을 포괄하고 있는 의미로 MS사에서는 'Any Time, Any Place, Any Device'라는 것을 모토로 하고, 'Microsoft.NET is the Microsoft XML Web Services platform'라고 정의하고 있다. 닷넷은 처음 등장할 때 자바를 흉내낸 언어라는 오명을 쓰기도 했지만, 핵심기술인 CLI(Common Language Infrastructure)와 C#언어가 2001년 12월 ECMA(European Computer Manufacturers Association) 국제 표준의 승인을 얻었고, IOS(International Organization for Standardization)에 표준승인을 받기 위해 노력중이다.

MS사에서 내 놓은 닷넷의 장점은, 닷넷을 기반으로 하는 웹 서비스는 표준 인터페이스를 사용하여 소프트웨어간의 통신을 간편화시키며 다른 응용프로그램과 쉽게 통합될 수 있는 환경을 제공한다는 것, 닷넷 프레임워크의 CLR을 사용하면 개발자는 거의 모든 프로그래밍 언어를 사용하여 웹서비스를 구현 할 수 있다는 것, 분사 시스템환경에서 어떤 위치의 웹 서비스의 웹서비스를 실행시키더라도 CPU의 사용률과 네트워크의 트래픽을 줄여 보다 향상된 기능을 제공한다는 것이다.

2) 닷넷 프레임워크

닷넷 프레임워크는 크게 클래스 라이브러리와 실행 환경의 두가지로 구성되는데, 닷넷에서 이용되는 웹 서비스와 CLR 등의 기능을 하는 추상적인 클래스들의 집합이라고 할 수 있다. 본 논문에서 기술하게 될 DOM과 SAX, XPath에 관련한 Class들이 모두 포함되어 있고, (그림 14)는 닷넷 프레임워크의 구성 요소를 나타낸 그림이다.



(그림 14) 닷넷 프레임워크의 구성 요소

(1) 실행 환경 (공통 언어 실행 : Common Language Runtime)

자바의 가상 머신(VM : Virtual Machine)과 같은 기능을 하는 것으로써, 공통 자료 시스템(CTS : Common Type System)과 공통 언어 규약(CLS : Common Language Specification) 등 기타 메타 데이터가 지켜야 할 기본적인 규약에 따라 JIT 컴파일, 가비지 컬렉션 등의 기능을 수행하는 실질적인 실행 환경이다. 자바의 VM과 다른 점은 멀티 패러다임 언어를 지원하는 점이다.

(2) 기반 클래스 라이브러리(Base Class Library)

C# 등의 응용 프로그램을 작성하기 위하여 제공되는 클래스들의 집합이라고 할 수 있다. 여러 개의 클래스들로 분류되어 지는데 이들 클래스들의 집합을 '네임스페이스(namespace)'라고 한다.

(3) ADO.NET과 XML

ADO.NET에는 System.Data와 System.Xml 네임스페이스가 있는데, System.Data 네임스페이스는 데이터베이스를 다루기 위한 클래스들로 구성되어 졌고, System.Xml 네임스페이스는 XML을 다루기 위한 클래스들로 구성되어졌다.

(4) ASP.NET

ASP.NET,은 웹 응용 프로그램과 웹 서비스를 작성하기 위한 클래스들로 구성되어졌다.

(5) Window Forms

윈도우 폼은 윈도우 응용 프로그램을 작성하기 위해 필요한 클래스들로 구성되어졌다.

Ⅲ. 시스템 성능 평가

1. 실험 환경

본 실험에서는 XML 문서에서의 검색 속도와 관계형 데이터 베이스 시스템에서의 검색 속도를 비교하기 위해 XML문서의 엘리먼트 개수를 다양하게 하고 또한 같은 내용의 데이터를 관계형 데이터 베이스 시스템에 저장하여 닷넷 프레임워크의 클래스를 이용하여 성능비교를 해 보겠다.

1) 데이터 생성을 위한 Testbed 생성 과정

본 논문에서는 제주도 지역의 인구 중에서 남녀의 직업(학생 포함)을 데이터로 하기 위해 임의로 생성하는데, 사용되는 필드는 ID, 성명, 주민번호, 성별, 우편번호, 주소, 직업 순으로 하였다.

기초 자료의 필드의 각각의 생성 방법은 다음과 같다.

- ID : 증가치를 1로 두고 생성하였다.
- 성명 : 우리나라에서 사용되는 성씨 154개(2자 포함)를 사용 빈도수가 높은 성에

가중치를 두어 저장하고, 이름으로 쓸 수 있는 글자 431개를 따로 저장한 후 성(한 글자 또는 두 글자)와 이름 두자를 랜덤하게 추출하여 결합하는 과정을 거쳐 생성하였다.

- 주민번호 : 주민번호 생성 방식을 사용하는데 남녀 중의 직업을 검색하기 위해 1987년생(17살)인 고등학교 1학년의 나이부터 시작한다.

- 성별 : 1과 2중 하나를 선택하게 하여 남녀를 구분한다.

- 우편번호와 주소 : 제주도의 우편번호와 주소(세부주소를 제외)를 저장한 파일 두 개에서 랜덤하게 생성된 하나의 숫자와 일치하는 곳의 우편번호와 주소를 얻어오는 방식으로 추출하고, 나머지 세부주소는 1~9999까지의 숫자에서 랜덤하게 추출하였다.

- 직업 : 제주도의 직업 분류를 저장한 파일에서 랜덤하게 추출하는 과정에서 주민번호가 87로 시작하는 경우(만 17살의 고등학생)까지만 생성하였는데, 그 이후에 태어난 남녀는 모두 학생이라고 가정하여 데이터에서 제외하기 위함이고, 남자의 경우에 군대를 필한 4년제 대학 재학생까지의 79년생(만 25살)까지는 학생으로 배정하고, 여자인 경우는 4년제 대학 재학생까지의 81생(만 23살)까지를 학생으로 배정하였으며, 그 이전에 태어난 남녀에는 직업 분류를 랜덤하게 배정하였다.

2) XML 문서의 구성

XML문서의 구성방식은 크게 두 가지로 볼 수 있다. 엘리먼트를 위주로 구성하는 방식과 어트리뷰트 위주로 구성하는 방법이 그것인데, XML 문서의 구성 방식은 자료의 성질과 사용자의 의도에 크게 관련된다. XML문서내의 하나의 레코드를 엘리먼트로만 구성할 수도 있고, 어트리뷰트로만 구성할 수도 있지만 보통의 경우는 엘리먼트를 위주로 하고 어트리뷰트를 적당히 혼합하여 구성한다. 본 논문에서는 위의 두가지 경우의 구성 방식의 성능 차이를 보이기 위해 똑같은 데이터에 대한 레코드를 엘리먼트만으로 혹은 어트리뷰트만으로 구성하여 보았다.

성능 평가를 위한 XML 문서는 C언어 프로그래밍을 통해 1000개의 엘리먼트로 구성한 것부터 75만개의 엘리먼트까지 개수를 다양하게 변화시키며 생성하였으며, 똑같은 데이터를 가지고 테스트하기 위해 MS-SQL 서버 2000에 위와 똑같은 내용을 가지는 테이블들을 생성하였다.

두 가지의 XML 파일의 예를 (그림 15)와 (그림 16)에서 확인할 수 있다.

```
<?xml version="1.0" encoding="EUC-KR" ?>
<직업정보>
  <개인정보 ID="1" 성명="김음맥" 주민번호="770112-2994383" 성별="여" 우편번호="690-040" 주소="제주시 용담동 890" 직업="실내건축기술자" />
    …
 <개인정보 ID="123" 성명="윤귀열" 주민번호="320302-2016550" 성별="여" 우편번호="690-011" 주소="제주시 일도1동 610" 직업="도시계획기술자" />
</직업정보>
```

(그림 15) 어트리뷰트만으로 구성한 XML파일

```
<?xml version="1.0" encoding="EUC-KR" ?>
<직업정보>
    <개인정보>
        <ID>1</ID>
        <성명>김음맥</성명>
        <주민번호>770112-2994383</주민번호>
        <성별>여</성별>
        <우편번호>690-040</우편번호>
        <주소>제주시 용담동 890</주소>
        <직업>실내건축기술자</직업>
    </개인정보>
    <개인정보>
        <ID>2</ID>
        <성명>박주활</성명>
                …
    </개인정보>
</직업정보>
```


(그림 16) 엘리먼트만으로 구성한 XML파일

위와 같이 XML 문서를 생성하기 위해 사용한 텍스트 파일을 테스트에 사용한 데이터 베이스에 저장할 때의 변환은 (표 5)와 같이 설정하였다.

구분	ID	성명	주민번호	성별	우편번호	주소	직업
자리수	자동증가	10자리	15자리	5자리	10자리	50자리	20자리
타입	int	char	varchar	char	varchar	varchar	varchar

(표 5) 데이터 베이스에서의 필드명과 타입

3) 생성된 데이터의 MS-SQL Server 2000의 DB테이블로의 이전방법

C언어로 생성한 데이터 파일(XML형태가 아닌 데이터 구분 기호를 Tab으로 지정한 Text파일)을 MS-SQL Server의 테이블로 이전시키기 위해서 DTS 가져오기/내보내기 마법사를 이용하였다.

이 때 이전 시에 텍스트 파일을 테스트에 사용한 데이터 베이스에 저장할 때의 변환은 (표 5)와 같이 설정하였다.

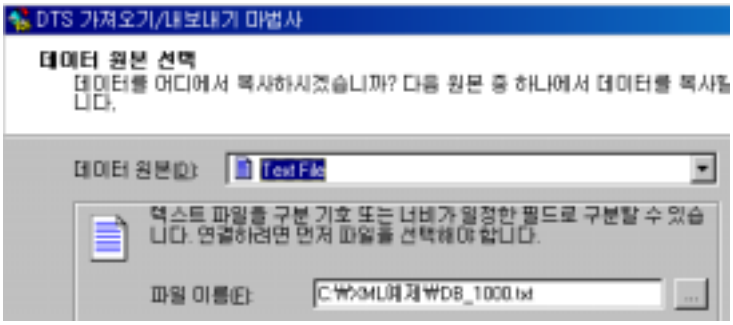
구분	ID	성명	주민번호	성별	우편번호	주소	직업
자리수	자동증가	10자리	15자리	5자리	10자리	50자리	20자리
타입	int	char	varchar	char	varchar	varchar	varchar

(표 6) 데이터 베이스에서의 필드명과 타입

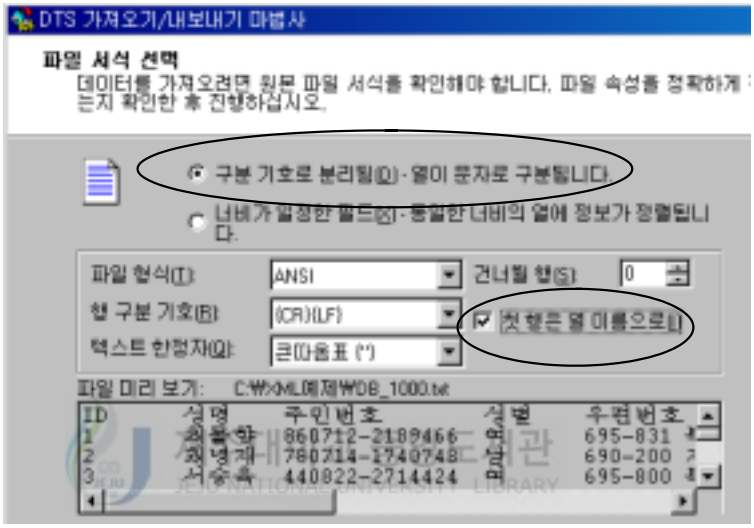
데이터 파일에서 MS-SQL Server로의 이전에 대한 실행 과정은 아래와 같다.

- MS-SQL Server의 [엔터프라이즈 관리자]실행 - 데이터베이스 선택 후 테이블에서 [데이타 가져오기](DTS Wizzard)를 실행한다.

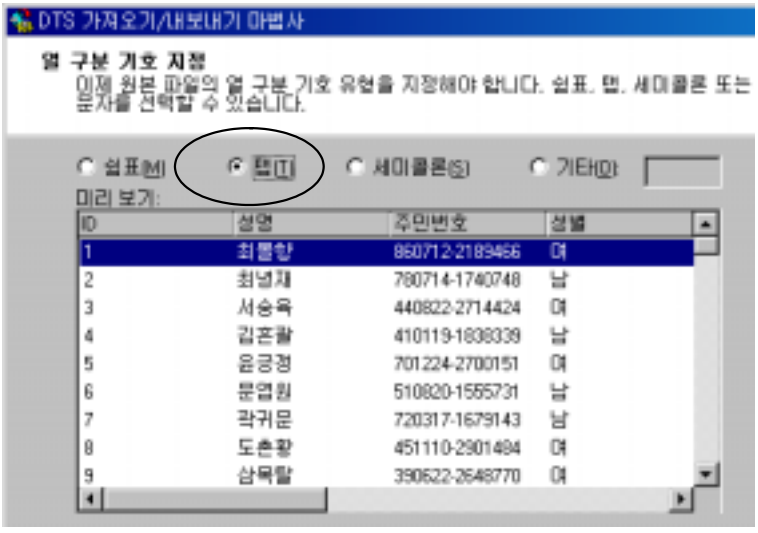
- 데이터 원본 선택 부분에서 데이터 원본을 Text File로 지정하고, 데이터 파일(Text)을 선택한다.



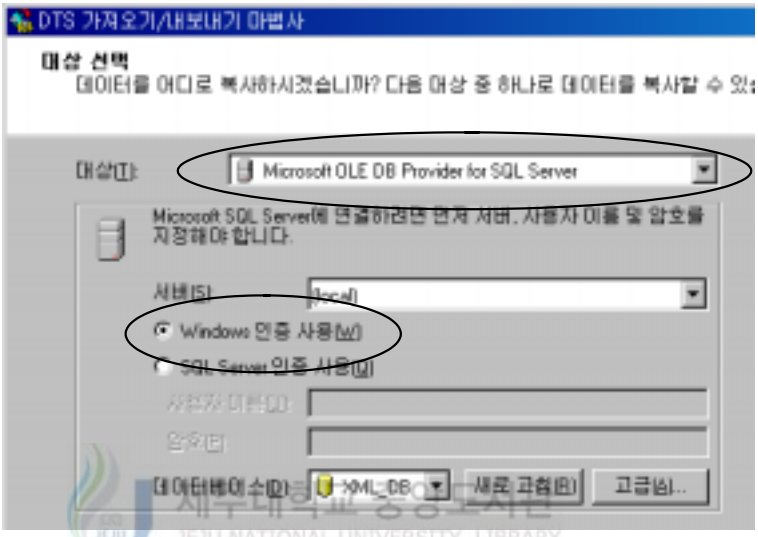
- 데이터 파일을 MS-SQL Server로 이전하기 위해 '구분 기호 분리됨'을 지정해 주고 '첫 행은 열 이름으로'에 체크를 해 준다. 첫 행을 열 이름으로 지정하는 경우는 데이터 파일의 첫 행의 Tab으로 구분된 내용이 테이블의 필드명으로 지정이 되도록 원하는 경우이다.



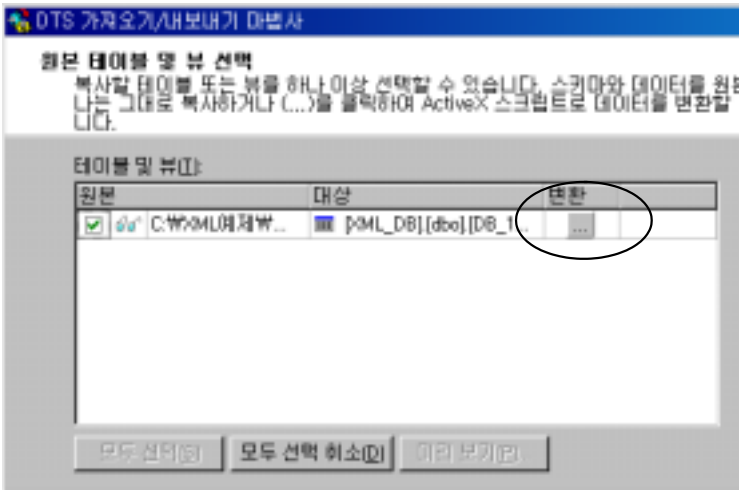
- 열 구분 기호를 지정해 주는데 이때 데이터 파일이 Tab으로 구분되어 있으므로 탭(T)를 선택해 준다.

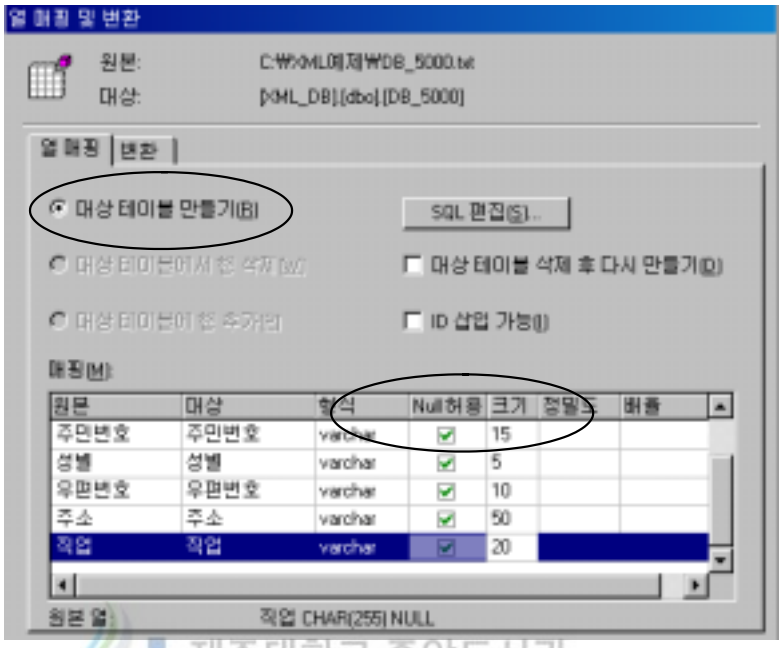


- 대상 선택을 해 주는데 MS-SQL Server의 OLE DB Provider를 선택해 주며, 이때 MS-SQL Server를 설치할 때의 데이터 베이스로의 접근 인증방법을 선택해 주어야 한다.

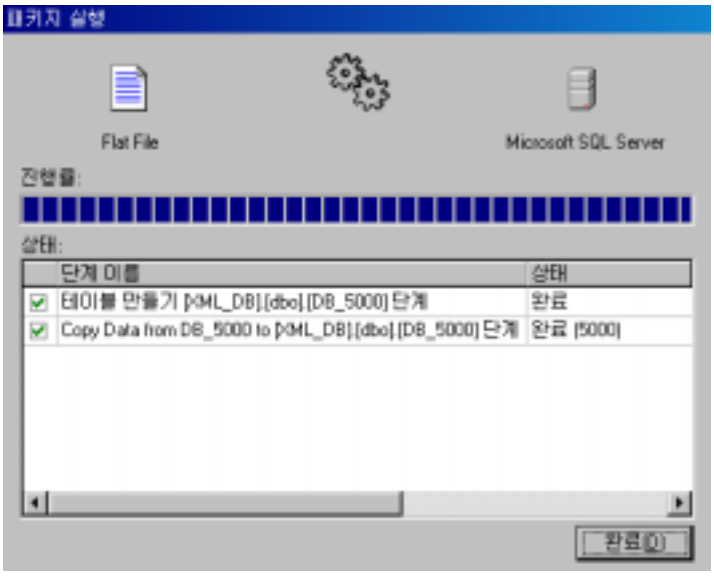


- 원본 테이블 및 뷰 선택이 나오는데, 이때 '변환'의 '열 매핑' 부분에서 테이블로 이전되는 데이터의 타입과 크기 등을 정해 줄 수도 있고, 대상 테이블의 생성 혹은 기존 테이블의 데이터를 삭제하면서 새로이 데이터를 이전시킬 수도 있다.





- 변환을 실행하면 테이블이 생성됨과 동시에 데이터 파일에서 MS-SQL Server로 데이터가 이전되는 것을 확인할 수 있다.



- 데이터 이전이 완료된 후 엔터프라이즈 관리자를 이용해 테이블이 제대로 생성

이 되었고, 각 필드명에 대해 데이터가 제대로 들어가 있음을 확인할 수 있다.

ID	성명	주민번호	성별	우편번호	주소	직업
1	정회점	870414-2619354	여	690-232	제주시 이호2동	학생
2	곽석사	780113-2148170	여	695-812	북제주군 조천읍	수의사
3	최여면	401201-2061191	여	690-062	제주시 화북2동	시스템엔지니어
4	박계탄	850625-1162677	남	690-190	제주시 외도동	학생
5	임걸십	650330-2809530	여	695-800	북제주군 구좌읍	전자택공학기술자
6	이중생	770922-1244226	남	697-011	서귀포시 중앙동	공인노무사
7	박익춘	790702-1642907	남	690-011	제주시 일도1동	학생
8	정잡골	810103-2809442	여	695-835	북제주군 한림읍	학생
9	한부영	521021-2281556	여	690-090	제주시 봉개동	여행안내원
10	강홍막	560103-1937903	남	697-130	서귀포시 색달동	악기제조원
11	강복은	441121-1214571	남	695-834	북제주군 한림읍	도자기제조원
12	윤당묘	300426-2164547	여	699-830	남제주군 대정읍	법률사무원

아래의 (그림 17)은 XML문서를 데이터베이스 시스템으로 이전한 경우의 경우를 도식화해 본 것이다.

```
<?xml version="1.0" encoding="EUC-KR" ?>
<직업정보>
  <개인정보 ID="1" 성명="정회점" 주민번호="870414-2619354" 성별="여" 우편번호="690-232" 주소="제주시 이동2동 783" 직업="학생" />
    …
  <개인정보 ID="5000" 성명="묵시준" 주민번호="600823-1437041" 성별="남" 우편번호="690-050" 주소="제주시 건입동 625" 직업="세라믹기술자" />
</직업정보>
```


ID	성명	주민번호	성별	우편번호	주소	직업
1	정회점	870414-2619354	여	690-232	제주시 이호2동 783	학생
2	곽석사	780113-2148170	여	695-812	북제주군 조천읍 와산리 1933	수의사
3	최여면	401201-2061191	여	690-062	제주시 화북2동 140	시스템엔지니어
4	박계탄	850625-1162677	남	690-190	제주시 외도동 8	학생
5	임걸십	650330-2809530	여	695-800	북제주군 구좌읍 765	전자택공학기술자
6	이중생	770922-1244226	남	697-011	서귀포시 중앙동 177	공인노무사
7	박익춘	790702-1642907	남	690-011	제주시 일도1동 1797	학생
8	정잡골	810103-2809442	여	695-835	북제주군 한림읍 한림리 1464	학생
9	한부영	521021-2281556	여	690-090	제주시 봉개동 918	여행안내원
10	강홍막	560103-1937903	남	697-130	서귀포시 색달동 13	악기제조원

(그림 17) XML문서와 DB테이블에서의 관계 도식

기존 연구에서 XML문서를 데이터베이스 시스템으로 이전하는 방식과 데이터베이스에 저장되어 있는 데이터를 XML문서로 변화시키는 방식이 있지만, 본 논문은 XML문

서 자체를 저장시스템으로 보는 경우와 데이터베이스에 같은 데이터가 존재하는 경우에 대해 닷넷프레임워크의 클래스를 이용하여 검색하는 성능평가가 주 목표이므로 기존 연구의 저장방식과 추출방식을 적용하지 않았음을 언급해 둔다.

4) 검색 조건과 검색 대상 파일들

2)에서 제시한 XML 문서의 검색 성능의 비교를 위해 검색 조건(표 6)을 다르게 해 보았는데, 엘리먼트 수를 1000개에서 75만개까지로 다양하게 구성된 XML 문서에서의 검색 결과는 검색 조건이 구체화될수록 검색 결과는 줄어드는 것을 볼 수 있고, 하나의 조건에 따라 검색 결과가 늘어나는 것을 볼 수 있다. 이것은 XML 문서를 구성하기 위한 데이터 파일의 생성 시에 데이터의 조합이 랜덤하게 구성되어졌음을 반증해 준다. 또한, 본 논문의 실험에서는 조건 1에 대해서 검색 시간만을 측정하여 비교 분석하였다.

	검색 요청문 내용
검색요청문1	직업 = '통계학자'
검색요청문2	성별='남' and 직업='통계학자'

(표 6) 검색 요청문의 내용

레코드수 / 요청문	1000개	1만	5만	10만	50만	75만
요청문1	6	36	177	392	1851	2843
요청문2	2	17	83	193	941	1370

(표 7) XML 문서에 대한 검색 조건과 결과(단위 : 개)

5) 실험환경

본 논문에서는 XML문서의 검색 성능의 비교를 위한 구성방식은 데이터를 엘리먼트로만 이루어져 있는 문서와 어트리뷰트로만 구성되어 있는 경우의 두 가지 방식으로 사용하였고, 이 문서의 각각에 대해 DOM방식으로 문서를 검색하는 방식과 SAX방식으로 문서를 검색하는 방식, XPath를 이용한 방식을 비교 분석하여 보고, 데이터 베이스 시스템에 저장되어 있는 똑같은 데이터를 닷넷 프레임워크의 ADO.NET을 이용해 검색

하는 방식을 서로 비교 분석하였다.

(1) 시스템 환경

XML 문서와 데이터 베이스 시스템에서의 검색 성능 비교를 위해 Windows 2000 Server(펜티엄 Ⅳ, CPU : 1.4GHz, RAM : 1GM, HDD : 40GB)에서 실험을 실시하였는데, C#언어의 지원을 위해 Visual Studio .Net을 설치하여 닷넷 프레임워크 SDK의 클래스 및 데이터 엑세스 컴포넌트를 이용할 수 있도록 하였고, 저장하부 구조인 데이터 베이스 시스템으로는 관계형 데이터 베이스인 마이크로 소프트의 MS-SQL Server 2000을 설치 운영하였다.

XML 문서의 파서는 마이크로 소프트 사에서 배포하고 있는 Microsoft XML Core Services(MSXML) 4.0을 이용하였는데, 이것은 XML 문서가 W3C의 XML 스키마(XSD)를 권장사항을 광범위하게 지원하도록 구성되었으며, SAX2(Simple API for XML)에 따라 순차적 XML 처리 아키텍처에 대한 지원이 확대되어진 것이다.

(2) 실험관련 클래스 및 파일

본 논문에서 사용된 XML 문서의 검색과 관련된 닷넷 프레임워크의 클래스를 구분한다면, DOM을 지원하는 클래스와 SAX를 지원하는 클래스, XPath를 지원하는 클래스, ADO.NET을 지원하는 클래스들로 크게 나눌 수 있다. DOM계열에는 XmlDocument, XmlDataDocument, XmlNode, XmlNodeList, XmlLinkedNode, XmlAttribute 등을 예로 들 수 있으며, SAX계열에는 XmlReader와 XmlWriter 등을 들 수 있는데, XmlWriter는 문서의 검색보다는 XML 문서를 생성하는데 유용하게 사용될 수 있다. XPath를 지원하는 클래스에는 XPathNavigator, XPathDocument, XPathExpression, XPathNodeIterator 등이 있으며, ADO.NET을 지원하는 클래스로는 DataSet, DataAdapter, DataTable, DataRow, DataColumn, XmlDataDocument, DataReader, Command 등을 들 수 있다. 실험에 사용된 XML 문서와 데이터베이스 시스템의 데이터 베이스에 대한 성능 평가는 DOM 방식, SAX 방식, XPath, ADO.NET을 이용한 성능 평가로 세부화하여 실시하였고, 이를 위한 닷넷 프레임워크의 클래스들을

(표 8)에서 요약 정리하였다.

방식	XML 문서의 구성방법	실험에 사용된 C# 파일이름	사용된 대표 클래스명
DOM	어트리뷰트로만 구성	XA-DOM	XmlDocument
	엘리먼트로만 구성	XE-DOM	XmlDocument
SAX	어트리뷰트로만 구성	XA-SAX	XmlTextReader
	엘리먼트로만 구성	XE-SAX	XmlTextReader
XPath	어트리뷰트로만 구성	XA-XPath	XPathDocument
	엘리먼트로만 구성	XE-XPath	XPathDocument
ADO.NET	데이터 베이스 테이블로 구성	ADO_DataSet	DataSet
	데이터 베이스 테이블로 구성	ADO_Reader	DataReader

(표 8) 실험에 사용된 XML 파일 내역 및 관련 클래스

(표 8)에서 '사용된 대표 클래스명'을 보면 하나의 클래스로 구성한 것을 볼 수 있는데, 여기서 제시한 클래스만이 아닌 다른 클래스를 더불어 사용하였으며, 실험의 명확성을 주기 위하여 각각의 실험에 필요한 최소한의 클래스들을 사용하였음을 언급한다. XA-DOM에서는 모든 어트리뷰트가 하나의 엘리먼트를 구성하여 하나의 레코드를 이루는 XML문서에서, 특정 조건에 따른 검색 결과를 얻기 위해 DOM과 관련된 XmlDocument 클래스를 이용하는데 XML문서를 메모리에 로딩한 후 XmlNode 클래스를 이용해서 검색을 했다. XE-DOM은 여러 개의 엘리먼트가 하나의 레코드로 이루어진 파일에 대한 DOM 관련 클래스를 이용한 것이며, XA-XPath와 XE-XPath는 XA-DOM과 XE-DOM에서와 동일한 XML 문서를 XmlDocument가 아닌 XPathDocument를 이용한 문서 로딩과 XPath관련 클래스를 사용해 XML 문서 내의 특정 조건에 맞는 레코드를 얻어오는 파일이다. XA-SAX와 XE-SAX는 SAX방식을 이용하는 검색 파일인데, SAX는 공식적인 문서 표준은 아니지만 닷넷 프레임워크에서는 XmlReader라는 클래스를 제공하여 SAX 방식에 대해 지원해주고 있다. SAX방식인 XmlWriter는 XML 문서내의 검색이 아닌 XML 문서의 생성에 관계된 클래스이므로 본 논문의 실험에서는 제외하였다.

ADO-DataSet과 ADO-Reader는 XML 문서의 검색과는 상관없이 단지 데이터 베이스 시스템의 데이터베이스에서 특정 조건에 맞는 결과를 얻어오기 위해 닷넷 프레임워

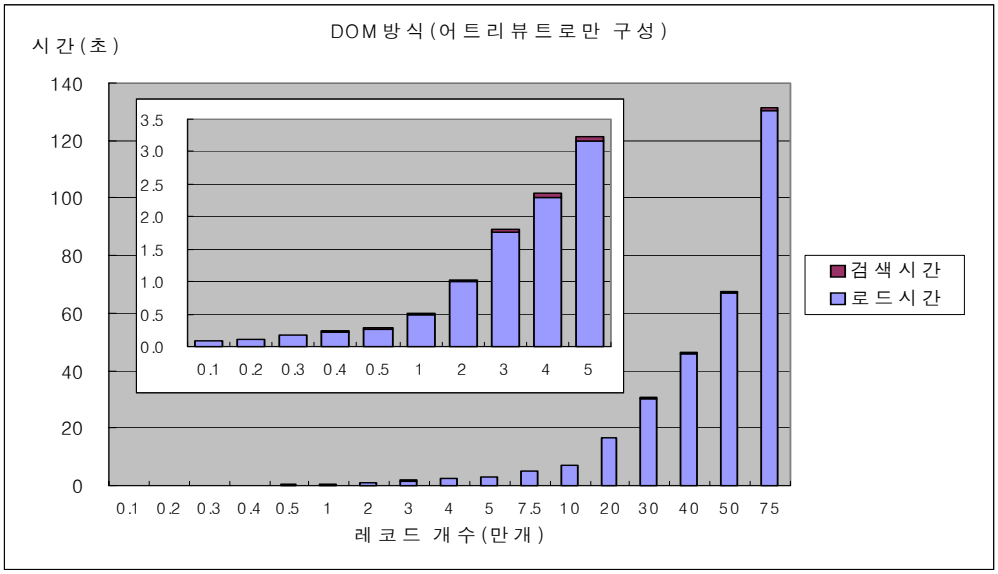
크의 ADO.NET을 이용하는 검색 파일이다. ADO.NET의 DataSet은 메모리 내에 특정 테이블과 동일한 형태의 가상테이블을 생성하고 데이터를 채워 넣은 상태에서 특정 조건에 맞는 검색을 하도록 지원하는 클래스로써 데이터 베이스에 다시 접속하지 않아도 되는 장점이 있고, DataReader는 메모리를 사용하지 않으면서 레코드를 검색하는 클래스로서 시스템에 부하를 줄여 줄 수 있지만 검색을 위해서 데이터 베이스와 계속적으로 연결이 지속되어야 하는 단점이 존재한다.

본 논문에서 언급되는 검색 시간들은 닷넷 프레임워크의 DateTime클래스의 속성인 Now를 이용하여 측정하였고, 각각의 성능 평가를 위한 파일들의 검색 시간은 10회 반복하여 측정한 값의 평균값으로 하였다. XML문서에 대한 검색 시간은 특정 루프를 통한 검색 요청문에 대한 결과값을 얻어 온 것이 아닌 문서의 로딩 등을 시작으로 출력까지의 시간을 모두 합한 시간을 말하는데, 결과값이 여러 개인 경우는 출력시간에 영향을 줄 수 있으므로 실제 측정 시에는 출력하는 부분을 주석 처리하여 제외시켰다. 데이터 베이스를 이용하게 되는 실험에서는 데이터 베이스에 접속하는 시간부터 결과값을 출력하는 시간까지의 시간을 검색시간으로 하였고, XML 문서에서의 검색 시간 측정과 동일하게 출력 부분은 주석으로 처리하여 제외시켰다.

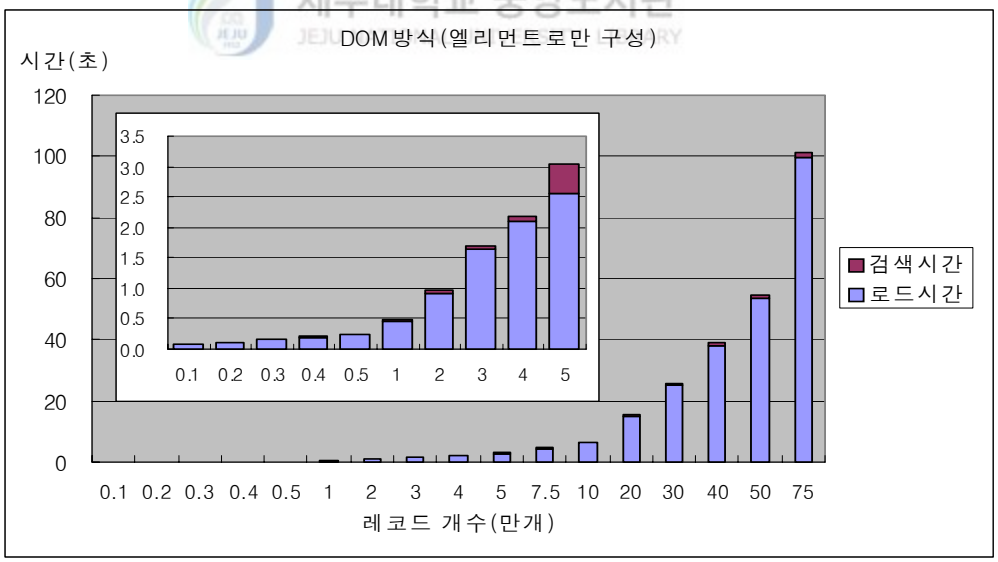
Ⅳ. 닷넷 프레임워크를 이용한 검색 성능 평가

검색 시간의 측정을 위해 각각의 C#파일에 닷넷 프레임워크의 클래스들을 이용하여 측정할 수 있도록 구성하였는데, 각 단계별로 시간이 얼마정도 걸리는지를 측정하여 어떤 부분이 검색 시간에 영향을 주는지를 먼저 확인해 볼 필요가 있기에 전체 검색 시간에 대한 각 단계별 소요시간을 측정하였다.

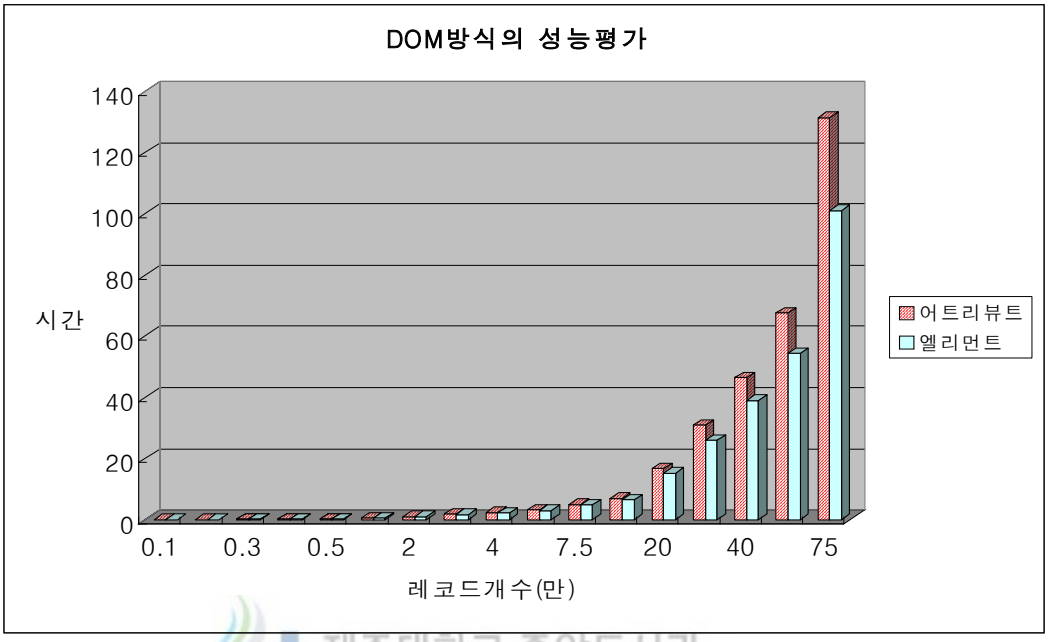
아래의 그림들은 각 방식별로 그와 관련된 클래스를 이용하여 각 단계를 그래프로 표시하여 본 것이다. 먼저, (그림 17)과 (그림 18)은 DOM방식으로 XML문서들에 대해 검색 요청문을 수행시킨 결과에 대한 그래프이다.



(그림 18) DOM방식으로 XML문서를 검색한 결과(XA-DOM파일)



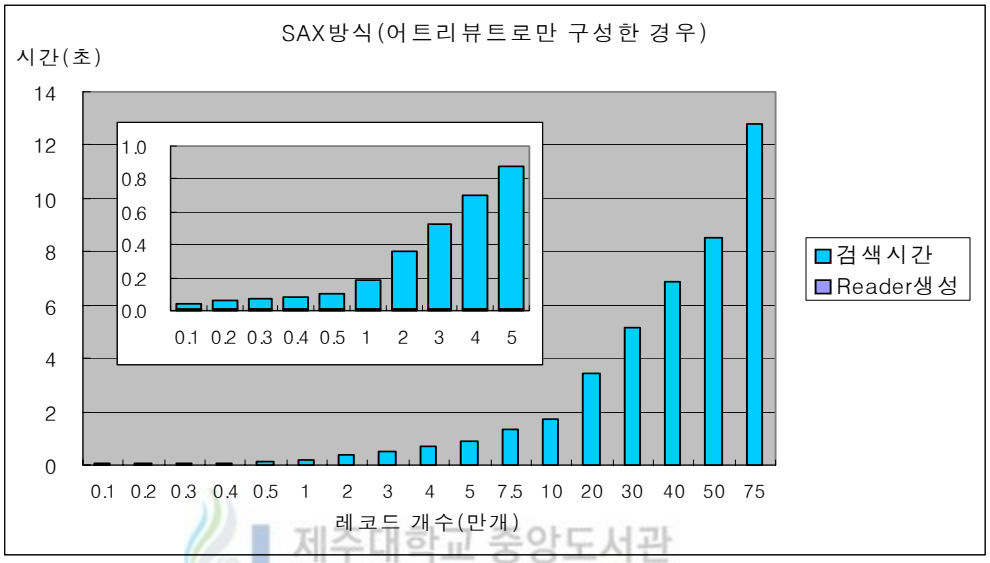
(그림 19) DOM방식으로 XML문서를 검색한 결과(XE-DOM파일)



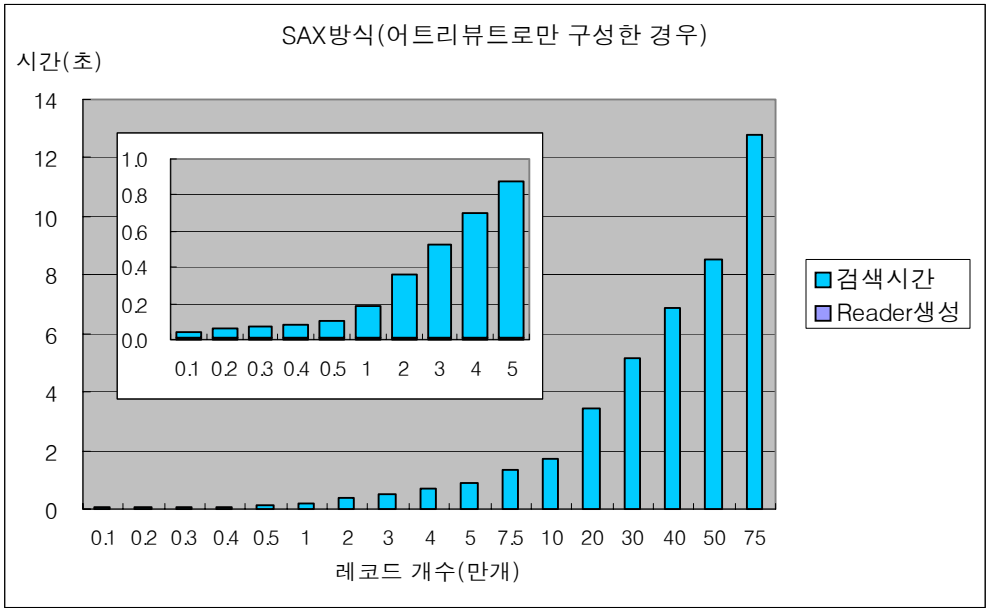
(그림 20) DOM방식으로 XML문서를 검색한 결과

위의 그림들은 XML의 구성을 어트리뷰트로만 혹은 엘리먼트로만 두 가지로 구성하고, 각각에 대해 DOM방식을 지원하는 클래스인 XmlDocument을 이용하여 XML문서를 로딩한 후 검색 요청문에 대해 단계별로 결과값을 얻어 온 것이다. 그림에서 보는 것처럼 DOM방식을 이용하는 경우는 검색 요청문에 대한 실제적인 검색 시간보다 XML문서를 로드하는데 대부분의 시간을 소비하고 있는 것을 볼 수 있다. 반대로 생각한다면 XML문서를 로드하는 시간을 무시할 수 있다면 상당히 작은 시간(75만개 레코드를 가지는 XML문서에 대한 검색요청문에 대한 시간은 1.5초를 넘지 않음)에 검색 결과를 얻어 낼 수 있는 장점이 있게 된다. 그리고, 4만개의 레코드를 가지는 경우까지는 전체 검색시간과 검색요청문에 대한 검색 시간이 거의 비슷하지만 5만개의 레코드를 가지는 경우부터는 XML문서를 어트리뷰트보다는 엘리먼트로 구성하는 경우가 75만개 레코드까지에서 20%~30%정도로 빠르다는 것을 확인할 수 있다. 즉, 5만개의 레코드를 가지기 전까지는 XML문서를 어떻게 구성하던지 거의 검색 속도의 차이를 느끼지 못하지만 그 이후부터는 되도록 엘리먼트로 구성하는 것이 DOM방식을 지원하는 클래스를 이용하는 경우의 검색에 유리함을 알 수 있다.

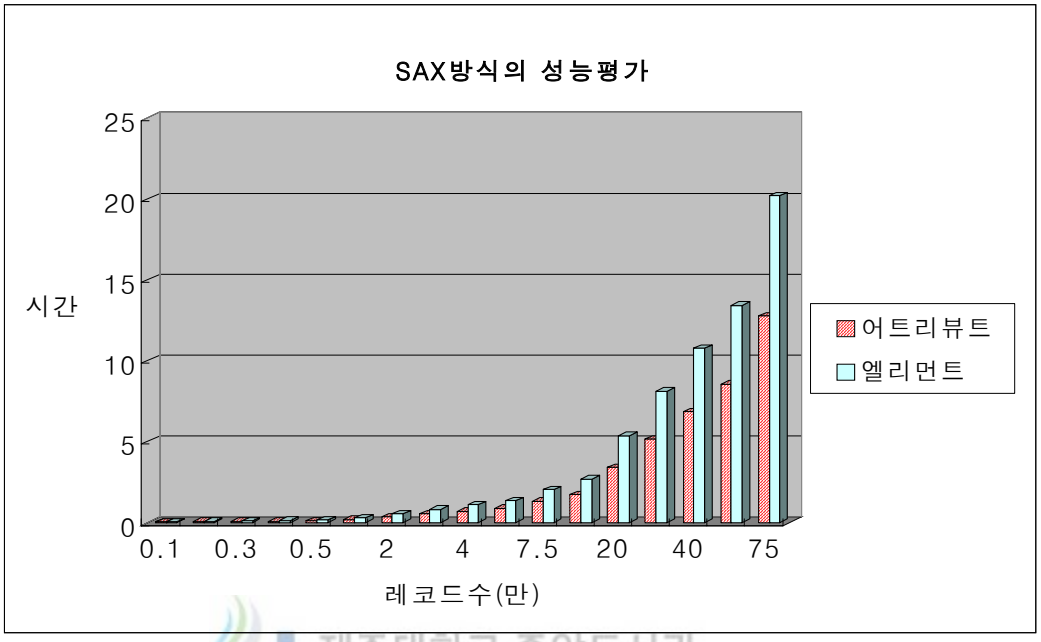
다음의 (그림 21)와 (그림 22)은 SAX방식을 이용하여 XML문서를 검색하여 얻어 온 결과이다.



(그림 21) SAX방식으로 XML문서를 검색한 결과(XA-SAX)



(그림 22) SAX방식으로 XML문서를 검색한 결과(XE-SAX)



(그림 23) SAX방식으로 XML문서를 검색한 결과

위의 그림들에서는 SAX방식을 지원하는 XmlReader클래스를 이용하여 XML문서를 검색한 결과인데, SAX방식의 XmlReader는 이벤트 드리븐 방식으로 문서에 대해 검색을 하게 되므로 문서를 로딩하는 과정이 빠지게 되고, 대신 Reader객체를 생성하는 시간을 추가하여 검색 요청문에 대한 전체 검색 시간을 측정하였다.

검색 시간의 전체 시간 중 Reader객체를 생성하는데는 0.01초도 안 걸리는 시간으로 그래프 상에선 눈으로 판별하기 어려운 것을 확인할 수 있고, 전체 검색 시간의 대부분의 시간을 검색 요청문에 대한 실제적인 검색 시간으로 소비되는 것을 알 수 있다.

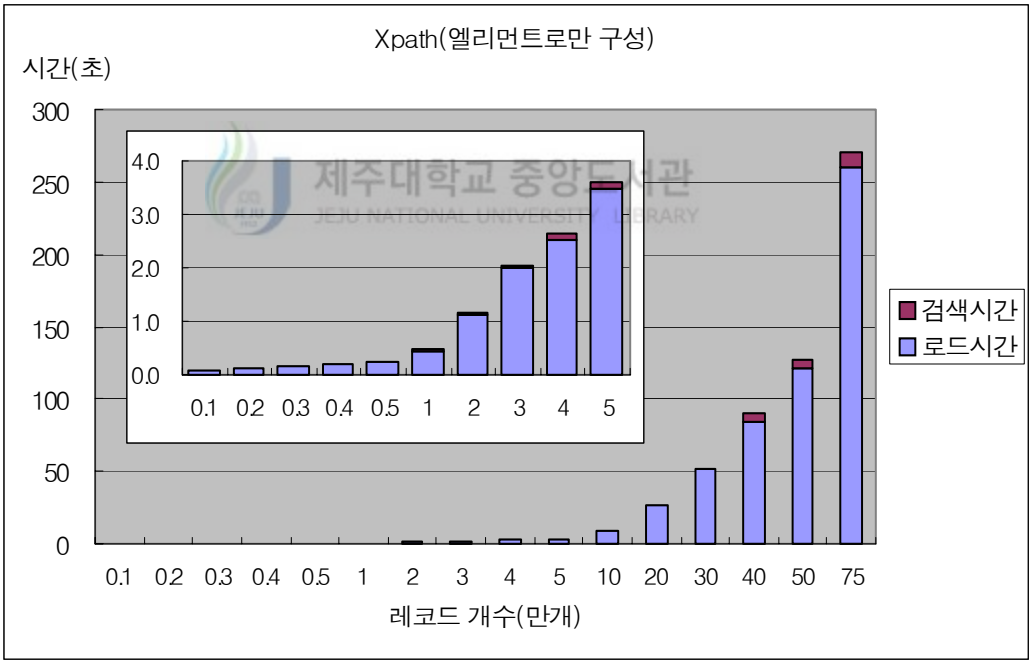
앞에서 실험으로 확인한 DOM방식과 비교해 볼 때 1/10의 시간으로 검색이 이루어짐을 볼 수 있는데 이것은 XML문서 자체를 저장시스템으로 사용한다는 가정을 하게 되면 검색 속도에서 8배 이상의 시간 절약을 가져 올 수 있게 된다.

그리고, SAX방식을 이용한 검색 속도 측정에서는 DOM방식과는 반대로 XML문서가 엘리먼트로 이루어진 경우보다 어트리뷰트로 이루어져 있는 경우가 검색 속도가 빠름(75만개 레코드인 경우 XA-SAX는 12.77초, XE-SAX는 20.19초)을 확인 할 수 있다. 하지만, 전체 검색 시간면에서 DOM방식을 이용해 측정한 경우와 비교해 볼 때는 현

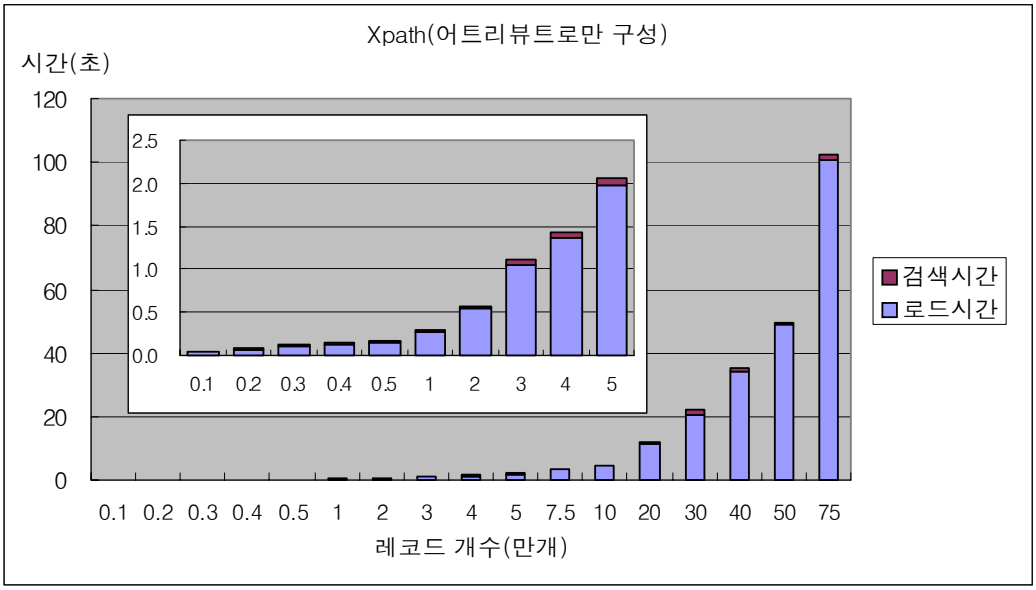
저하게 빠른 검색 시간으로 결과를 얻어 오는 것을 볼 수 있지만 실제적인 검색 요청문에 대한 검색 시간만을 비교해 볼 때는 DOM방식을 이용한 경우가 현저하게 빠르게 검색 결과를 보여준다는 것을 볼 수 있다.

즉, DOM방식을 사용하여 검색 결과를 얻어 오기 위한 경우에는 XML문서를 로딩하는 시간이 많이 걸린다는 약점이 보여 전체 검색시간이 길어지기는 하지만 실제적인 검색 요청문에 대한 결과를 얻어 오는 과정에서의 검색 시간은 이벤트 드리븐 방식인 SAX방식보다 더 빠른 결과를 얻어내는 것을 볼 수 있다.

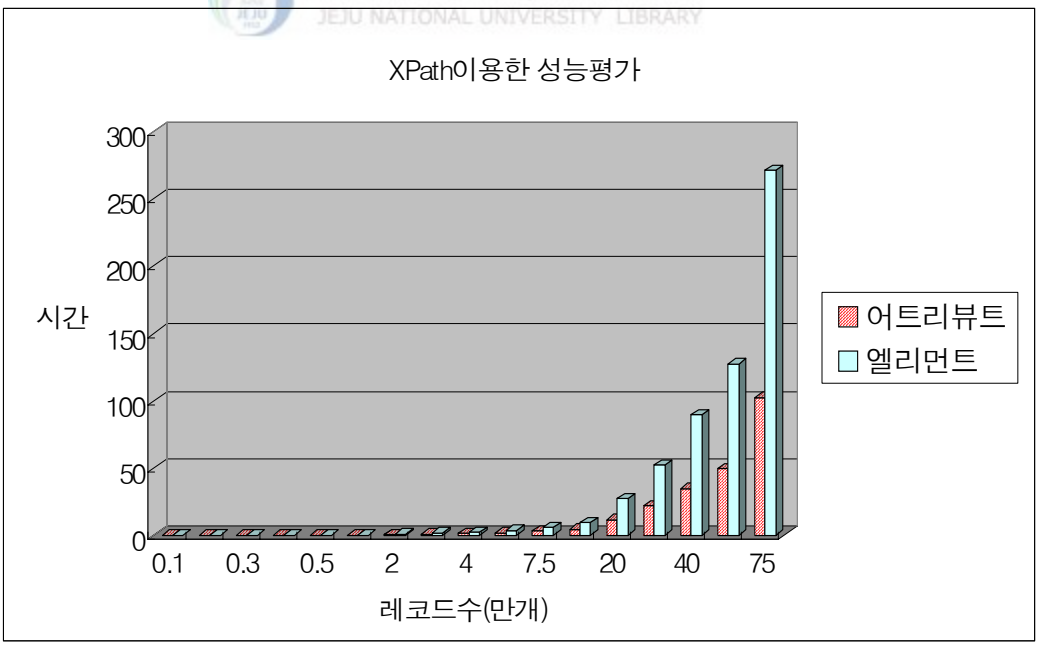
다음의 (그림 24)과 (그림 25)는 XPath를 지원하는 클래스인 XPathDocument를 이용한 경우의 검색 시간 결과의 그림이다.



(그림 24) XPath를 이용해 XML문서를 검색한 결과(XE-XPath)



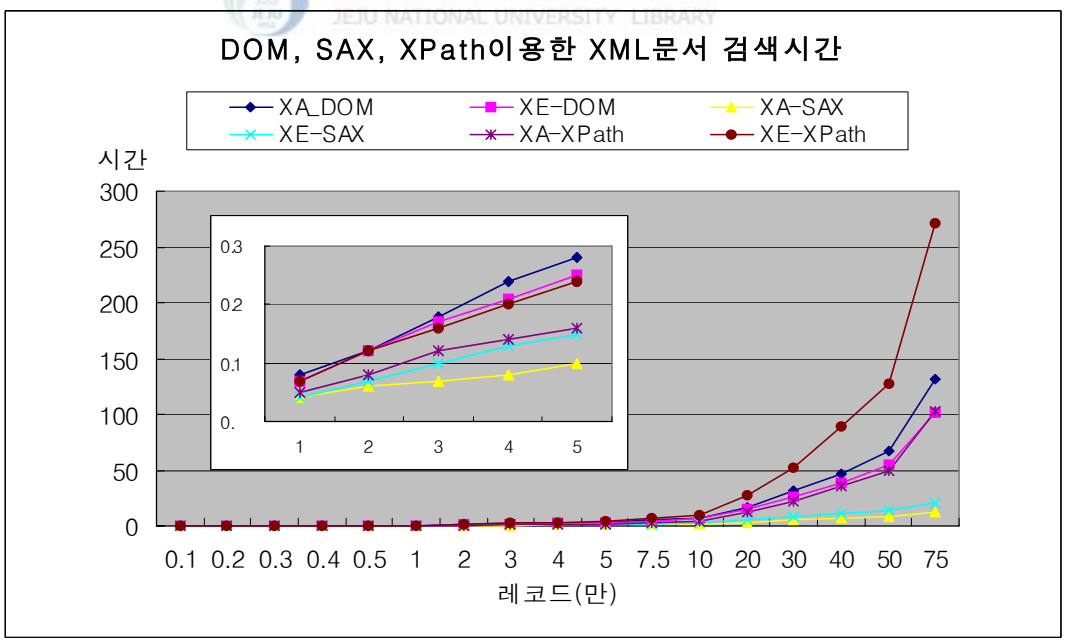
(그림 25) XPath를 이용해 XML문서를 검색한 결과(XA-XPath)



(그림 26) XPath를 이용해 XML문서를 검색한 결과

그림들에서 보는 것과 같이 XPath를 이용하여 검색하는 경우는 DOM을 이용하는 경우와 비슷한 결과가 보여지는 것을 알 수 있는데, XPathDocument 클래스가 DOM방식의 XmlDocument처럼 메모리에 XML문서를 로딩하기 때문이다. DOM방식과 비교를 해 볼 때, 엘리먼트로 구성된 경우에는 XPath를 이용한 경우가 훨씬 많은 시간이 로드시간에 소비되는 것에 비해, 어트리뷰트로만 구성된 경우에는 XPath를 이용한 경우가 로드시간이 더 적게 걸리는 것을 볼 수 있고, 검색 요청문에 대한 검색 속도는 DOM에 비해 XPath를 이용하는 경우가 XML문서를 엘리먼트 혹은 어트리뷰트로 구성한 경우에 모든 면에서 떨어짐을 볼 수 있다. 그리고, XPath를 이용한 경우만 볼 때, 앞의 실험과 마찬가지로 엘리먼트로 이루어지는 경우보다 어트리뷰트로 이루어지는 경우가 XML문서를 로딩하는 시간과 검색 요청문에 대한 결과를 얻어내는 시간이 2배 이상 적게 걸리는 것을 확인 할 수 있다.

다음의 (그림 27)은 XML문서에 대해서 DOM방식과 SAX방식, XPath를 이용해서 검색 요청문에 대해서 결과를 얻어온 그래프를 종합해 본 것이다.



(그림 27) 각 방식에 따른 XML문서에서의 검색 시간

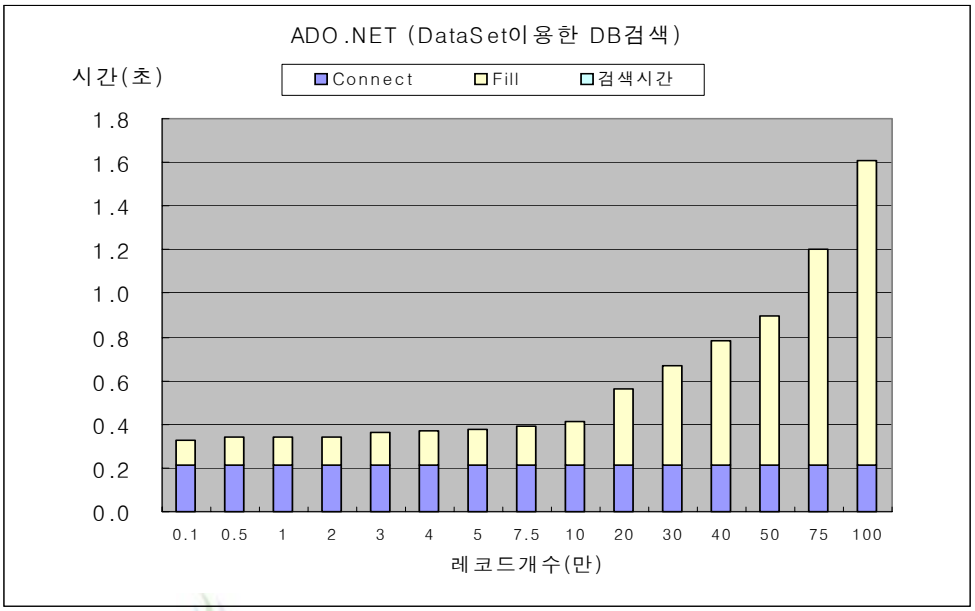
Element수	XA_DOM	XE_DOM	XA_SAX	XE_SAX	XA-Xpath	XE-Xpath
0.1	0.08	0.07	0.04	0.04	0.05	0.07
0.2	0.12	0.12	0.06	0.07	0.08	0.12
0.3	0.18	0.17	0.07	0.1	0.12	0.16
0.4	0.24	0.21	0.08	0.13	0.14	0.2
0.5	0.28	0.25	0.1	0.15	0.16	0.24
1	0.52	0.48	0.19	0.29	0.29	0.48
2	1.03	0.96	0.36	0.55	0.57	1.18
3	1.81	1.7	0.53	0.83	1.11	2.06
4	2.36	2.19	0.7	1.09	1.43	2.63
5	3.24	3.03	0.88	1.36	2.05	3.61

(표 9) 각 방식에 따른 XML문서에서의 검색 시간(5만개까지인 경우)

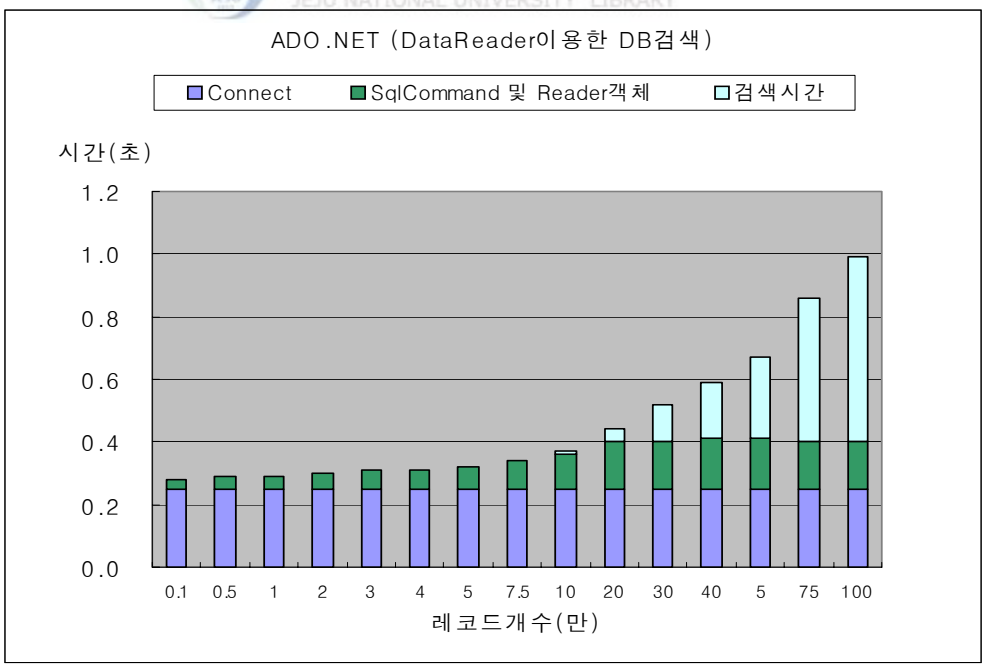
그림에서 보는 것과 같이 DOM방식의 XML문서의 검색 시간과 XPath를 이용한 검색시간보다는 SAX방식의 XML문서의 검색 시간이 빠르다는 것을 볼 수 있는데 레코드 개수가 4만개 이상인 경우부터는 어트리뷰트로 구성되는 경우 2배 이상의 차이를 보이는 것을 알 수 있다. 즉, XML문서에서 직접적으로 검색 요청문을 수행할 때는 DOM방식과 XPath방식다는 SAX방식이 상대적으로 빠른 결과를 얻어 올 수 있으며, (그림 19)와 (그림 20)에서 보는 것처럼 SAX방식을 채택해 검색 수행을 시키는 경우에는 XML문서의 문서 구성이 엘리먼트보다는 어트리뷰트로 구성된 것이 좀더 빠른 수행 결과를 얻어 올 수 있음을 알 수 있다.

다음의 (그림 28)과 (그림 29)는 관계형 데이터 베이스인 MS-SQL Server 2000에 XML문서와 동일한 데이터를 1. 3)에서 보인 방법을 통해 테이블로 이전시킨 후 ADO.NET을 지원하는 클래스들을 이용하여 검색 요청문을 실행한 결과이다.

(그림 28)에서 실험한 것은 DataSet을 이용해 데이터베이스의 실제적인 검색 성능을 알아보기 위한 것인데, DB에 Connect하고 난 후 모든 데이터를 DataSet이란 가상 메모리 상에 올린 후 특정 레코드를 검색하는 것이 아니라 DB자체에 대해 검색 요청문에 해당하는 결과를 얻어오는데 걸리는 시간을 측정한 것이다. 이런 이유로 DataSet을 이용한 검색에서는 검색시간이 0초에 가까운 시간이 측정되어 그림 상에 나타나지 않았음을 언급해 둔다. DataReader를 이용한 것도 위와 유사하게 측정하였지만, 스트림에서 검색 요청문에 대한 검색을 수행하므로 검색 시간이 존재하게 된다.



(그림 28) ADO.NET을 이용한 DB에 대한 검색 결과(ADO_Reader)

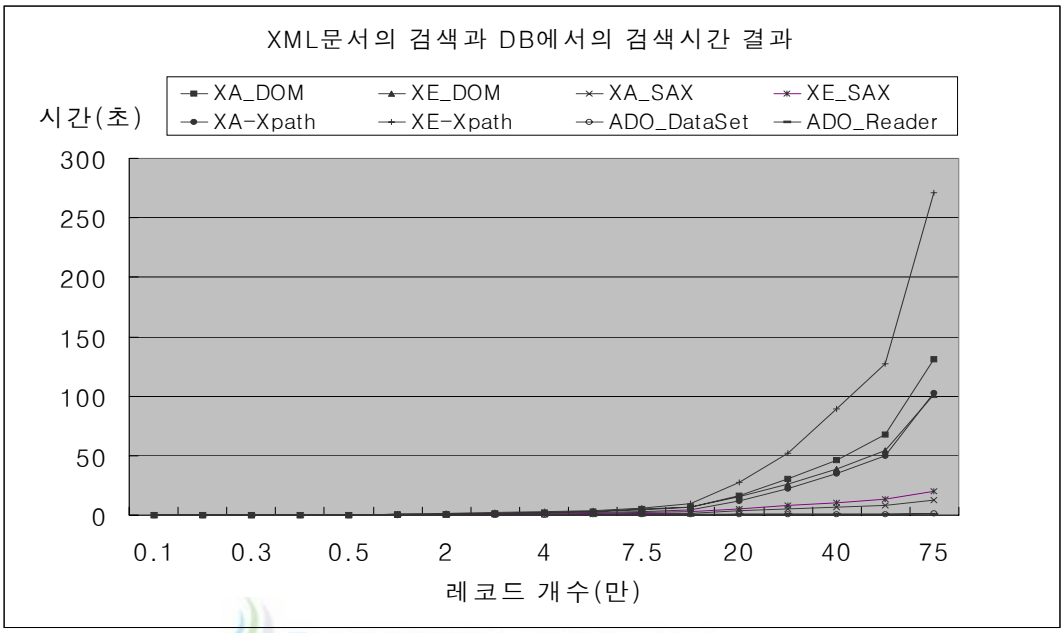


(그림 29) ADO.NET을 이용한 DB에 대한 검색 결과(ADO_DataSet)

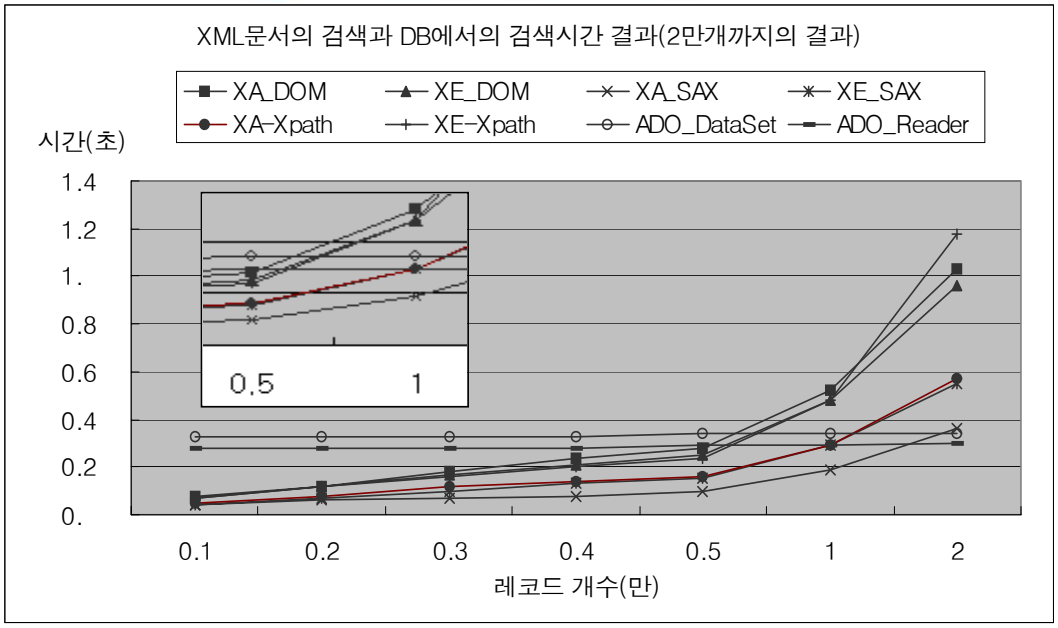
위의 두 개의 그림에서 DataSet과 DataReader의 두 경우 모두 데이터 베이스에 Connect하는데 일정시간을 소비하는 것을 볼 수 있다. DataSet을 이용한 경우는 그 외에 Fill(데이타 베이스의 실제 테이블에서 가상 테이블인 DataSet에 데이터를 이전시키는 과정)과정에 대부분의 시간(1.40초)을 소비하며 검색 시간은 0초에 가까운 결과를 보여 그래프에서 나타나지 않음을 언급해 둔다.

DataReader를 이용한 경우는 Connect하는 시간 이외에 SqlCommand객체와 Reader 객체를 생성하는 시간과 검색 요청문에 대한 결과를 얻어내는 시간에 전체 검색 시간을 소비하는 것을 볼 수 있는데, DataSet이 가상의 테이블을 메모리에 생성하는 것과는 다르게 DataReader는 데이터 베이스의 테이블에 있는 데이터를 스트림으로 얻어 오면서 검색 요청문에 해당하는 데이터를 얻어 오기 때문에 그 만큼의 검색 시간이 증가함을 확인할 수 있다. 하지만, 이 또한 XML문서를 직접적으로 읽어오는 시간과 비교해 본다면 굉장히 작은 속도로 볼 수 있다.

다음의 (그림 30)는 XML문서를 DOM과 SAX, XPath를 이용해 검색하는 시간과 데이터베이스에 저장된 똑같은 데이터를 DataSet과 DataReader를 이용해서 검색하는 시간을 총괄한 그래프이고, (그림 30)은 (그림 29)의 성능비교 결과에 대해 좀더 자세한 이해를 돕기 위해 2만개까지의 엘리먼트를 가지는 경우에 대해서 추출한 그래프이다.



(그림 30) 레코드 수에 따른 XML과 데이터베이스 성능 비교 결과



(그림 31) 2만개까지의 레코드 수에 따른 결과

(그림 30)과 (그림 31)에서 보는 것과 같이 레코드 개수가 7천 5백개 이하인 경우에는 닷넷 프레임워크의 ADO.NET 즉, 데이터 베이스 시스템을 저장 시스템으로 이용한 경우보다 XML문서를 저장 시스템으로 이용하여 검색을 하는 것이 빠른 결과를 보이는 것을 볼 수 있고, SAX를 이용한 경우에는 어트리뷰트로만 구성된다면 1만 5천개일 경우까지도 빠른 결과를 확인할 수 있다. 하지만 2만개 이후부터는 차이를 보이기 시작함을 확인 할 수 있다.

또한, DOM방식과 SAX방식, XPath만을 비교해 볼 때 어트리뷰트로 구성된 XML을 이용하는 경우 SAX방식이 4만개의 레코드를 가지는 경우부터 XML문서에 대한 검색 시간이 다른 것에 비해 2배 이상 좋은 결과를 보여줌을 알 수 있다. 또한, XML문서의 구성방식에서는 엘리먼트로 구성된 것보다는 어트리뷰트로 구성되어 있는 것이 검색 시간을 줄일 수 있다는 것을 알 수 있다.

Ⅳ. 결 론

본 논문에서는 XML문서에 대해서 검색, 삽입, 삭제 등을 가능하게 해 주는 API인 DOM과 SAX, 그리고, 트리 탐색 언어인 XPath, ADO.NET을 이용하여 특정 요청문에 대한 검색 성능에 대해 실험을 했으며, 각각의 검색은 닷넷 프레임워크에 기반한 클래스들을 이용하여 프로그래밍을 하였다. 제주도내의 남녀(학생 포함)의 직업에 대한 Testbed를 생성한 후에 XML문서와 MS-SQL Server 2000 내에 엘리먼트로만 혹은 어트리뷰트로만 구성하여 검색 성능을 비교 분석하였다.

본 논문의 실험 결과를 볼 때, 닷넷 프레임워크의 클래스들을 이용하는 경우 데이터베이스 시스템을 저장시스템으로 구성하여 검색을 수행하는 것이 가장 좋은 방식임을 알 수 있었고, XML문서를 저장 시스템으로 이용하는 경우에는 SAX방식을 지원하는 닷넷 프레임워크의 클래스를 이용하는 것이 검색시간에 대한 좋은 결과를 얻어 올 수 있으며, 어트리뷰트로 XML문서를 구성하는 경우에는 4만개 레코드 이후부터는 2배 이상의 성능이 좋아짐을 확인 할 수 있었다. 또한, DOM방식을 이용하는 경우에는 XML문서의 구성 방식에서는 레코드를 어트리뷰트보다는 엘리먼트로 구성하는 것이 검색 성능에 효율적이고, SAX방식과 XPath방식, ADO.NET을 이용하는 경우에는 엘리먼트보다는 어트리뷰트로 구성하는 것이 검색하는데 소비하는 시간을 줄일 수 있음을 알 수 있다. 또한, 레코드 개수가 만개 이하인 경우에는 데이터 베이스 시스템을 저장 시스템으로 이용하는 것보다 XML문서를 저장시스템으로 이용하는 것이 더 좋은 검색 성능을 가져 올 수 있다.

이 후부터의 연구는 기존 실험에서 확장해 검색 성능에 미치는 하드웨어적인 면, 즉 CPU의 차이, 메모리의 차이 등의 변수를 두고 검색성능에 미치는 영향을 비교분석해 볼 것이며, 또한, 현재 이슈가 되고 있는 XQL이라는 질의방식을 이용해 XML문서를 검색하는 방식에 대해 연구를 진행해 볼 것이다.

V. 참고 문헌

Christophides, V., Abiteboul, S., Cluet, S., and Scholl, M. 1994. From Structured Documents to Novel Query Facilities. In Proc. *Int'l Conf. on Mangement of Data.* ACM SIGMOD. Minneapolis, Minnesota. pp313~324

D. Florescu and D. Kossman. 1999. Storing and Querying XML Data Using an RDBMS. IEEE Data Engineering Bulleltin. 22(3). pp24~27

J. Shanmugasundaram, et al. 1999. Relational Databases for Querying XML Documents : Limitataions and Opportunities. Proc. of the 25th VLDB conf.

DOM(Document Object Model). 1999. "DOM Level 2 Specification : W3C Working Draft", http://www.w3.org/TR/1999/WD-DOM-Level-2-19990923/

J.Clark and S. DeRose. 1999. "XPath 1.0 : XML Path Language". W3C Recommendation. Available at http://www.w3.org/tr/xpath/

J. McHugh, S. Abiteboul, R. Goldman, D. Quassand, J. widom. Lore. 1997. A Database Management System for Semistructured Data. SIGMOD Record. 26(3). pp54-66

SAX(Simple API for XML). 1998. Available at http://www.megginson.com/SAX/

W3C(World Wide Web Consortium). 1996. Available at http://www.w3.org/

XML(Extensible Markup Language). Available at http://www.w3.org/XML/

강미경, 박소정. 2003. XML문서 검색과 DB를 이용한 검색의 성능 비교. 제주대학교 기초과학연구소. 제16권, 1호, p73~87

김경원, 손기락. 2002. XPath 처리를 위한 효과적인 XML 저장 시스템의 설계 및 구현. 정보과학회 2002년 추계학술대회. VOL.29 NO.02

김노환. 2001. XML DOM을 이용한 웹문서 검색 알고리즘. 컴퓨터산업교육기술학회 논문지. VOL.02 NO.06 pp.0775 ~ 0782

김노환, 정충교. 2001. XPath 질의를 이용한 DB2XML 알고리즘 설계 및 구현. 컴퓨터산업교육기술학회 논문지. VOL.02 NO.06 pp. 0837~0844

김훈, 홍의경. 2000. 객체 관계 데이터베이스를 이용한 XML문서 저장 모델 설계. 한국정보과학회 2000년 추계학술대회 논문집. Vol.27 NO.02. pp225~227

선승상, 박상윤, 엄영익. 2000. XML 문서의 객체지향적 관리를 위한 XML DOM 소프트웨어의 설계 및 구현. 정보처리학회 2000년 춘계학술대회. VOL.07 NO.01

신인혜, 선경희, 강순철. 2003. XML문서의 검색 성능 평가. 제주대학교 기초과학연구소. 제16권, 1호. p89~97

이강찬, 손호, 박기석. 2001. XML 표준화 동향. 한국 정보 과학회지. VOL.19 NO.01 p.0006~0014.

이강찬. 2001. DOM (Document Object Model) 기술 동향 분석서 http://dblab.ce.cnu.ac.kr/~dolphin/xml/atoz/dom.html

이기훈, 한욱선, 김민수, 이종학, 황규영. 2003. ODYSSEUS/XMLStore: 오디세우스 객체관계형 DBMS를 위한 XML 저장 시스템

이용석. 손기락. 1998. XML문서 저장 시스템 설계 및 구현. 정보과학회 학술발표 논문집

정희경. 1999. 차세대 웹 문서 표준 XML. 한국정보 처리 학회지. VOL.06 NO.03 p.0025~0035

정태선, 박상원, 한상영, 김형주. 2002. XML 데이터를 위한 객체지향 데이터베이스 스키마 및 질의 처리. 정보과학회 논문지. VOL.29 NO. 2. pp.0089~0098