# XML을 RDBMS에 저장하기 위한 Analyzer 설계 및 구현

정민경° 홍동권 남재열
계명대학교 정보통신학부 컴퓨터공학과
{skallet6050°}@hotmail.com

# Design and Implementation of XML Analyzer on RDBMS

Minkyoung Jung° D.K. Hong J.Y. Nam
Keimyoung University. The Department of Computer Engineering

요 약

오늘날 XML이 디지털 정보교환의 표준으로 자리잡은 후 XML문서를 데이터베이스에 저장하고 원하는 정보를 효율적으로 질의하기 위한 연구가 활발히 진행되고 있다. 특히 XML질의어를 RDBMS 상에서 처리하기 위해 그리고 XML문서정보를 정확하게 추출하여 효율적으로 관리하기 위해 다양한 기법을 동원한 XML 인덱스 Table의 연구가 계속 되고 있다. 하지만 아무리 설계가 잘 된 XML인덱스 table이라 할지라도 이에 저장될 정보들을 XML문서로부터 빠른 시간에 파싱하여 관계형 테이블에 로드하기가 쉽지 않다. 이에 본 논문에서는 RDMS환경에서 XML문서를 관리하고 질의를 처리할 수 있도록 XML 인덱스 table을 디자인 하였으며 이에 정확한 값이 좋은 성능을 가지면서 저장되도록 XML문서의 데이터 정보를 추출하는 XML Analyzer를 설계 및 구현하였다. 우선 Analyzer를 구현하기 위해서는 XML Parser를 사용해야 되는데 본 장에서는 이벤트 기반 방식인 SAX를 통해 XML문서를 파싱하여 데이터를 추출하고 그 결과값을 RDMS상의 XML 인덱스 Table에 저장한다. 마지막으로 이를 실제 구현하고 Test한 내용을 근거로 하여 본 장에서 소개하는 XML Analyzer가 다른 방식보다 성능면에서 훨씬 우수하다라는 사실을 입증한다.

## 1. 서 론

최근 인터넷의 발달은 디지털 정보교환의 표준으로 확고한 자리를 굳힌 XML의 사용을 가속화 시키고 있다. 또한 XML 데이터 양이 기하급수적으로 증가함으로써 보다 효율적으로 XML데이타를 저장하고 질의하기 위한 연구가 필요하게 되었고 이에 다양한 XML전용 데이터베이스가 출현하게 되었다. 하지만 현재 출시된 XML전용 데이터베이스 시스템은 그 성능이 확실하게 검증된 바가 없으며 20년이상 거듭 발전 해온 관계형 데이터베이스 시스템을 따라오지 못하고 있다. 따라서 본 논문에서는 관계형 데이터베이스 시스템 환경하에 XML문서를 저장하고 질의를 처리할 수 있도록 XML인덱스 테이블을 디자인하고 이에 저장될 정보를 추출하는 XML Analyzer를 설계 및 구현한다.

우선 RDBMS상에서 디자인된 XML 인덱스 테이블에 정확한 XML문서 정보가 삽입되려면 XML문서를 파싱하는 XML Parser가 필요하다. 대표적으로 객체 모델 기반방식의 DOM API와 이벤트 기반 방식의 SAX API가 있다. DOM은 먼저 XML document를 검색하고 분석한 뒤 이를 트리형태로 메인메모리상에 올려놓는다. 이는 XML문서가 클 경우 실행 속도가 느려지게 하는 주요 요인이 된다. 하지만 사용하기가 쉬우며 특정 엘리먼트로의 접근이 가능하고 편집 및 삭제가 가능하다. 반면에 SAX는 XML문서를 처음부터 읽어 나가면서 이벤트가 발생할 때마다 특정작업을 수행한다. 그러므로 실행 속도는 DOM보다 빠르지만 삽입과 삭제가 어려운 단점이 있다. 본 장에서는 XML문서의 내용을 변경하기 위한 것이 아니라 필요한 정보를 효과적으로 추출하여 XML 인덱스 테이블에 저장하기 위한 것을 목적으로 하기 때문에 SAX API를 이용하여 XML Analyzer를 구현한다.

## 2. XML Analyzer의 설계

RDBMS상에서 XML문서에 대한 질의했을 때 원하는 결과를 얻기 위해서는 XML문서로부터 엘리먼트, 어트리뷰트, 각 경로, 문서내의 키워드 정보 이렇게 네 종류의 정보를 추출하여 아래에 나오는 인덱스 테이블에 삽입하는 XML Analyzer를 이용한다. 만약 사용자로부터 질의가 주어졌다면 아래에 나오는 인덱스 테이블을 질의하여 원하는 결과값을 찾는다.

```
<Inproceeding mdata="2002" key="conf/wg/GurskiW00">
  <author>Frank Gurski</author>
  <title>The Tree-Width of Clique-Width Bounded Graphs Without
   <i>K <sub>n </sub></i>
   test
  </title>
</inproceeding>
```

<dblp.xml문서의 일부분>

아래의 테이블은 XML문서내에 존재하는 경로에 대한 정보를 추출한다

| docid | pathid | path | depth | path_cnt |
|---|---|---|---|---|
| 1 | 1 | Inproceeding | 1 | 1 |
| 1 | 2 | Inproceeding~/athor | 2 | 1 |
| 1 | 3 | Inproceeding~/title | 2 | 1 |
| 1 | 4 | Inproceeding~/title~/i | 3 | 1 |

*본 연구는 산업자원부 지역연구개발클러스터구축사업 중 경북대학교 첨단/예측 의료기술 클러스터 사업단의 연구비 지원을 받아 수행되었음

다음은 XML문서의 엘리먼트에 대한 정보를 추출한 테이블이다..

| id | eid | name | sibord | pathid | key_c | pid | info | dewey_n | value |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Inproceeding | 1 | 1 | | | 1 | #1# | |
| 1 | 2 | author | 1 | 2 | 2 | 1 | 2 | #1#1# | Frank.. |
| 1 | 3 | title | 1 | 3 | 8 | 2 | 3 | #1#2# | The Tree.. |
| 1 | 3 | title | 1 | 3 | 1 | 2 | 4 | #1#2# | test |
| 1 | 4 | i | 1 | 4 | 1 | 3 | 4 | #1#2#1# | K |
| 1 | 5 | sub | 1 | 5 | 1 | 4 | 5 | #1#2#1#1# | n |

다음은 XML문서의 Text에서 불용어를 제외한 주요 키워드들만 추출한 테이블이다.

| word | position | depth | id | eid | pathid |
|---|---|---|---|---|---|
| Frank | 1 | 3 | 1 | 2 | 2 |
| Gurski | 2 | 3 | 1 | 2 | 2 |
| Tree | 1 | 4 | 1 | 3 | 3 |
| Width | 2 | 4 | 1 | 3 | 3 |
| Clique | 3 | 4 | 1 | 3 | 3 |
| Bounded | 4 | 4 | 1 | 3 | 3 |
| : | | : | : | : | : |

다음은 XML문서의 어트리뷰트 정보를 추출한 테이블이다

| id | eid | aid | aname | avalue |
|---|---|---|---|---|
| 1 | 1 | 1 | mdata | 2002 |
| 1 | 1 | 2 | key | conf/wg/GurskiW00 |

위와 같이 4개의 각 인덱스 table에 XML문서로부터 추출된 정보가 저장되며 저장 될 때는 오라클에서 지원하는 SQL*LOADER를 사용한다. 이렇게 본 장에서 소개하는 XML Analyzer를 통해 RDBMS상의 인덱스 table에 XML문서 정보가 삽입되며 이를 검색함으로써 사용자가 입력한 질의를 정확하게 처리할 수 있다.

## 3. XML Analyzer의 알고리즘 및 구현

본 장에서 소개하는 Analyzer는 이벤트 기반 방식의 SAX를 사용한다. 보통 XML문서에서 발생하는 이벤트는 시작 엘리먼트, Attribute, contents, 종료 엘리먼트 크게 4가지로 볼 수 있다. 즉 문서의 처음부터 마지막까지 읽으면서 위의 이벤트가 발생할 때마다 본 논문에서 소개하는 알고리즘을 통해 정보를 추출하고 이를 XML 인덱스 table에 저장한다. 여기서 정보를 추출하는 도중 벡터와 해쉬 테이블이라는 자료구조를 이용하게 되는데 이들은 XML문서에서 발생한 경로를 가진다.

2장에서 소개한 dblp문서를 예로 든다면 다음과 같다. 우선 Location table에 저장 될 정보는 Inproceeding 시작 엘리먼트를 만날 경우 벡터를 확인하고 이와 동일한 경로가 없으면 벡터와 해쉬 테이블에 Inproceeding 경로를 추가한다. 만약 발생한 경로가 벡터에 있는 경우 해쉬테이블에 그 경로의 횟수를 1증가 시키면서 저장한다. 그리고 Inproceeding 엘리먼트의 정보는 stack에 저장하고 그대신 attribute정보는 Attribute 테이블에 저장한다. 그런 다음 author엘리먼트를 만나면 위와 동일한 방법으로 처리하되 contents는 stack에 저장한다. 만약 author의 종료엘레먼트를 만나게 되면 stack을 pop하여 엘리먼트의 내용과 키워드에 관한 내용을 저장한다.

지금까지 살펴본 내용을 바탕으로 본 장에서 소개하고자 하는 XML Analyzer의 알고리즘은 다음과 같다.

<XML Analyzer의 알고리즘>

```
[startelement]
if(시작엘리먼트가 연속적으로 나왔을 경우){
 부모 element의 character정보들을 관련스택에 push하는 메소드호출
parents 스택에 부모 element 정보를 push함
}else if(시작엘리먼트, character가 나온 다음 시작 엘리먼트를 만났을
경우){
 부모 element의 character정보들을 관련스택에 push하는 메소드를
호출
 부모 element의 character정보들을 파일에 적음
 parents 스택에 부모 element 정보를 push함
}else if(종료엘리먼트, character가 나온다음 시작엘리먼트를 만났을
경우){
parents 스택에 저장되어 있는 top정보와 현재 character정보를 파일
에 적음}
부모 엘리먼트 경로를 저장하는 메소드 호출()
현재 경로를 저장하는 메소드 호출()
attribute 정보를 저장하기 위한 메소드 호출()
[element]
if(종료 엘리먼트가 연속적으로 나왔을 경우){
 해쉬 테이블에 저장되어 있는 부모경로의 횟수를 0으로 만듬.
 parents 스택 정보를 pop
 contents 스택의 키워드 정보를 파일에 적음
 if(contents 스택의 top에 있는 value값이 있을 경우){
  if(방금 character을 읽었을 경우){
 parents 스택, 관련스택 정보를 pop, 현재 character와 같이 파일에
적음}
}else{
 parents 스택의 top element와 관련 스택내용을 pop해서 파일에 적
음}
}else{ element에 대한 정보를 파일에 적음
       키워드에 대한 정보를 파일에 적음}
[character]
if(종료엘리먼트를 만난 후 character를 만난 경우){
 contents 스택의 top에 현재 contents정보를 연결하는 메소드 호출}
```

위의 알고리즘을 구현한 XML Analyzer를 통해 XML문서로부터 데이터 정보를 추출하고 이를 RDBMS상의 인덱스 테이블에 저장함으로써 주어진 질의를 처리한다.

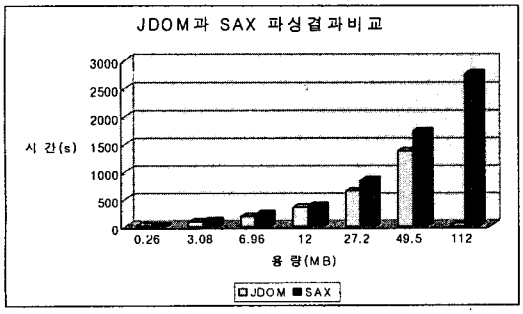
## 4. 성능평가 및 결론

본 장에서 제안하는 XML Analyzer를 구현한 환경은 팬티엄4 2.66GHz, 메인메모리 512MB, 운영체제는 윈도우 2000 Professional이며 JDBC와 jdk1.4.05를 이용하여 구현하였다. 성능평가는 XML Analyzer와 Jdom이 주어진 XML문서를 파싱하는데 소요한 시간을 각각 측정하였으며 추출한 데이터 정보를 실제 인덱스 table에 저장하는데까지 걸리는 시간을 다양한 입력방법별로 측정하였다.

우선 성능평가에 이용되는 XML문서는 다음과 같다.

| 종류 | 타입 | 문서의 개수 | 전체크기(MB) | 평균깊이 |
|---|---|---|---|---|
| Shakespeare | Real | 37 | 7.53 | 5.95 |
| DBLP | Real | 1 | 50.3 | 4.00 |
| Auction | Syntheti | 1 | 115.5 | 7.02 |

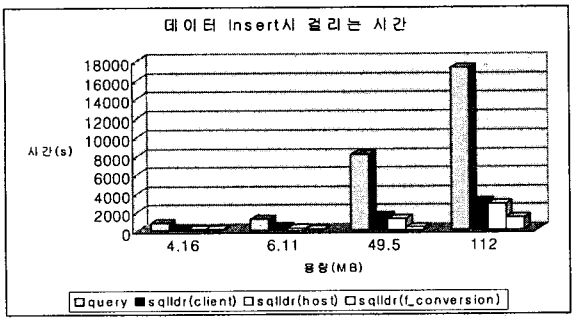
첫 번째 성능평가는 SAX API를 이용해 구현한 XML Analyzer와 JDOM의 XML 파싱성능을 비교하는 것이다. 결과는 다음과 같다.




<표1-파서성능비교>

위의 결과와 같이 JDOM을 이용할 경우 적은량의 XML 문서는 조금 더 빠른 속도로 파싱하지만, XML데이터의 크기가 커질수록 메모리 사용량이 커짐으로서 시스템이 다운되는 현상을 초래한다. 이와는 대조적으로 SAX를 이용하여 구현한 Analyzer는 문서의 크기가 커지더라도 어느 정도의 안정성을 보장하면서 파싱을 할 수 있다.

두 번째 성능평가는 파싱한 정보를 실제 인덱스 Tabledp 로드하는 시간을 비교한 것으로 오라클 9i에서 행해졌으며 이것은 리눅스 2.6.10, 팬티엄3 1.2GHz 듀얼CPU, 메인메모리1.5GB에 설치되어있다. 클라이언트는 위에서 살펴본 환경과 동일하다. 데이터를 테이블에 로드하는 방법으로 각각 Client에서 Query를 이용하는 방법, Client 그리고 Host에서 SQL*Loader를 이용하여 데이터를 데이터베이스의 Temp테이블에 로드한 다음 데이터 정보 변경 후 실제 인덱스 table로 로드하는 방법, 그리고 본장에서 소개하는 XML Analyzer의 데이터를 변경한 후 데이터베이스에 로드하는 방법의 성능을 비교하였다.




<표2-관계형 데이터베이스에 Insert하는 시간>

지금까지 살펴본 성능평가를 근거로 XML문서를 파싱하는 면에서나 파싱한 결과를 실제 인덱스 Table에 Inser하는 면에서나 본 장에서 소개한 XML Analyzer가 빠른 속도와 안정성을 보장하고 있다.

다음은 XML Analyzer가 파싱된 정보를 로드하는데 소요한 시간을 인덱스 테이블별로 비교분석한 결과표이다.

<테이블별 데이터 로드시간>

| 파일명 | 문서 개수 | 전체 크기 | loca-tion | element | attribute | word | total |
|---|---|---|---|---|---|---|---|
| shakes peare | 30 | 6.11MB | 1345 | 144,404 | 0 | 454,020 | 599,799 |
| dblp | 1 | 49.5MB | 78 | 1,201,512 | 265,893 | 2,176,995 | 3,644,479 |
| auction | 1 | 112MB | 514 | 1,863,366 | 381,675 | 10,286,464 | 12,532,020 |

위의 결과표를 본다면 데이터를 로드하는 시간이 Word 테이블에 편중되어 있음을 알 수있다. 이는 XML문서의 Text정보가 많고 이를 구성하는 Keyword의 개수도 많이 때문이다. 여기서 Word테이블은 특정 Keyword검색에만 주로 이용되므로 XML문서의 Text가 그리 길지 않고 특정 Keyword에 관한 질의가 많이 이루어지지 않는다면 이를 제외 하는 것이 바람직 할 것이다.

또한 여기서 그치지 않고 더욱 효율적으로 XML문서를 파싱하고 빠른시간내에 RDBMS 테이블에 저장할 수 있는 다양한 기법을 동원한 XML Analyzer가 개발 및 구현되어야 할 것이다.

## 참고문헌

[1]Yoshikawa, M., Amagasa, T,. "XRel: path- based approach to storage and retrival of XML Documents using relation database" 2001

[2]Igor Tatarinoy, Stratis D. Yiglas, Kevin B eyer, Javavel Shanmugasundaram, Eugene Shekita, Chun Zhang "Storing and Querying Orderd XML Using a Relational Database System"2002

[3]David Dehaan, David Toman, Mariano P., Consens, M.Tamer Ozsu "A Comprehensive XAuery to SQL Translation using Dynamic Interval Encoding" 2003

[4]"XML path Language (XPath) Version 1.0" http://w3c.org/TR/xpath 2005년 2월 검색