

XML 데이터 처리를 위한 XML-QL to SQL 번역기

(An XML-QL to SQL Translator for Processing XML Data)

장 경자[†] 이기호^{††}

(Gyeong-Ja Jang) (Ki-Ho Lee)

요약 XML이 다양한 유형의 웹 데이터를 구성하고 교환하기 위한 국제적인 표준으로 제안되었다. 다양한 애플리케이션에서 필요로 하는 데이터가 XML 문서에 저장되어 있을 때, 그 데이터를 검색하는 것은 중요한 일이다. 본 논문에서는 XML 문서를 관계형 데이터베이스 시스템에 저장하는 방법과 저장된 XML 데이터를 XML-QL로 질의하여 검색하는 방법을 제안한다. 즉, XML의 저장 방법을 제안하고 저장된 XML 데이터를 검색하기 위해 XML-QL로 들어온 질의를 관계형 데이터베이스 시스템의 전용 질의 언어인 SQL로 변환시켜 주는 번역기를 설계 및 구현한다. 본 논문의 의의는 번역기에 대한 설계와 구현을 자세히 기술하므로 번역기의 가능성을 보여주고 XML 질의의 포괄적인 분류와 그에 대한 SQL 관계형 질의로의 매펑을 보여주고 있다는 점이다.

키워드 : XML, XML 질의어, 관계형 질의어, 번역기, XML 데이터 처리, 질의 유형, 문서형 정의

Abstract XML has been proposed as an international standard for organizing and exchanging a great diversity of the Web data. It is important to retrieve components of stored XML documents that are needed by a wide variety of applications. In this paper, we suggest a method to store XML documents and to retrieve an XML data. In other words, we suggest the method of retrieving XML data is using XML-QL. So we need to mapping XML-QL queries to SQL queries. In this paper, we describe the design and implementation of an XML-QL to SQL translator on top of an RDBMS. The contributions of this paper include, besides the detailed design and implementation of the translator, demonstration of feasibility of such a translator, and a comprehensive classification of XML queries and their mappings to SQL relational queries.

Key words : XML, XML-QL, SQL, translator, processing for XML data, Query Type, DTD

1. 서 론

인터넷과 웹 기술 사용의 확산은 관계형 데이터베이스의 테이블과 같은 구조적인 데이터(structured data)[1]와 HTML과 XML 페이지와 같은 반구조적인 데이터(semi-structured data)[1] 그리고 이미지, 비디오 클립, 텍스트 문서와 같은 비구조적인 데이터(unstructured data)[2] 등 다양한 웹 데이터를 야기시켰다. 따라서 웹 데이터를 구성하는 다른 유형의 데이터를 통합해야 한 필요성이 제기되었고 XML이 웹 데이터를 구성하고 교환하기 위한 국제적인 표준으로 제안되었다[3]. 다양한

애플리케이션에서 필요로 하는 데이터가 XML 문서에 저장되어 있을 때, 그 데이터를 검색하는 것은 중요한 일이다. XML 문서를 저장하고 검색하기 위한 두 가지 방법이 있다. 첫째, 관계형 데이터베이스 시스템과 같은 기존의 데이터베이스 관리 시스템을 사용하는 방법[1, 4,5]과 XML 전용서버를 사용하는 방법[1,6]이 있다. 이 두 가지 경우에 모두 장단점이 있다. 첫 번째 경우, 사용자들에게 일반적으로 사용되고 있지만 이 시스템에서 다루고자 하는 XML 문서가 기존에 처리하던 데이터와 데이터 모델이 다르다. 따라서 데이터의 손실 없이 저장하기 위한 연구가 많이 이뤄지고 있다[1,5,7]. 두 번째 경우, XML 문서와 데이터 모델이 비슷하기 때문에 XML 문서에 포함된 데이터를 검색하는 방법이 비슷하지만 시스템의 사용이 제한적이다[1,6]. 따라서 본 연구에서는 가장 일반적으로 사용되고 있는 관계형 데이터베이스 시스템에 XML이 저장되었을 때, XML 데이터

[†] ㈔회원 : 삼성종합기술원 국제표준화팀 연구원

gjang@sait.samsung.co.kr

^{††} 총신회원 : 이화여자대학교 컴퓨터학과 교수

khlee@ewha.ac.kr

논문접수 : 2000년 9월 14일

실사완료 : 2001년 10월 8일

를 효과적으로 검색할 수 있는 방법을 제시한다. 즉, 관계형 데이터베이스 시스템에 XML-QL로 절의하여 XML 데이터를 검색하는 방법을 제안한다. 이를 위해 본 연구에서는 XML 데이터를 효과적으로 검색하기 위한 질의 유형을 분류하고 XML-QL로 들어온 질의를 관계형 데이터베이스 시스템의 SQL 질의로 변환시킨다. 그리고 나서 얻어진 결과를 XML 문서로 변환시켜 주는 번역기를 구현한다.

2. XML-QL 유형 분류

저장된 XML 문서에 포함된 데이터를 효과적으로 검색하는 방법은 중요하다. 그래서 XML 문서에 포함된 데이터를 검색하기 위해 W3C에서 제안한 XML-QL[8]의 질의 특징을 몇 가지 질의 유형으로 분류하고 관계형에 저장된 XML 데이터를 검색하는데 적용하고자 한다.

우선, XML 질의를 분류하기 위해 기준이 필요하다. 본 논문에서는 다음 세 가지 기준으로 표 1과 같이 8가지 유형으로 질의를 분류한다. 첫째, XML 문서 구조에 따라 분류한다. 즉, 한 개의 XML 문서에 포함된 데이터를 검색하는데 필요한 질의를 단일 문서 유형(Single Document Type)으로 분류하고 2개 이상의 XML 문서에 포함된 데이터를 추출하기 위해 필요한 질의를 다중 문서 유형(Multiple Documents Type)으로 분류한다. 둘째, 추출하고자 하는 데이터가 다른 데이터에 참조된 상태에 따라 분류한다. 즉, 검색하고자 하는 데이터(예를 들어, 애트리뷰트)가 다른 데이터에 참조될 경우, 링크(Link)를 사용해서 그 데이터를 검색할 수 있다. 이 경우는 단일 문서 유형과 다중 문서 유형에 모두 적용될 수 있다. 이를 단일 문서 유형에서는 링크(또는 참조)(Link 또는 Reference)로 다중 문서 유형에서는 명백한 링크(Explicit Link)로 구별한다. 셋째, 추출한 결과를 좀 더 의미 있게 재처리하기 위해 연산과 같은 후처리가 필요한 질의(Query Requiring Post-processing)와 후처리가 필요 없는 질의(Query Requiring Non-post-processing)로 분류한다. 이것도 단일 문서와 다중 문서 유형 모두에 적용된다. 위와 같은 세 가지 기준에 따라 표 1과 같이 8가지 유형의 질의로 분류한다.

표 1과 같이 후처리가 필요로 하지 않는 질의(Query Requiring Non-post-processing)는 프레디كت을 기본으로 하는 질의(Predicate Based Query)와 트래버서를 기본으로 하는 질의(Traversal Based Query)로 구분할 수 있다. 프레디كت을 기본으로 하는 질의(Predicate Based Query)는 프레디كت을 만족하는 질의이다. 예를

표 1 XML-QL 질의의 8가지 유형

XML Structure Link Based, Post processing	Single Document Type	Multiple Document Type
Query Requiring Non-post-processing	Predicate Based Query	Predicate Based Query
	Traversal Based Query	Traversal Based Query
Link Based Query	Link(Reference)	Explicit Link
Query requiring Post-processing	Sort	Grouping (Aggregation)

들면, 엘리먼트 노드의 값이 문자열과 일치한 엘리먼트를 찾는 질의가 있다. 트래버서를 기본으로 하는 질의(Traversal Based Query)는 루트 엘리먼트 노드에서부터 경로를 따라 항해하는 방식의 질의이다. 예를 들면, 상위 엘리먼트 이름이 무엇이든지 상관없이 엘리먼트 이름이 Lastname이거나 In인 엘리먼트를 찾는 질의가 있다. 프레디كت을 기본으로 하는 질의와 트래버서를 기본으로 하는 질의는 단일 문서 유형과 다중 문서 유형 모두 가능하다. 링크를 기본으로 하는 질의(Link Based Query)는 검색하려는 데이터가 다른 데이터에 참조되어 있는 경우에 링크를 사용해서 질의하는 것으로 단일 문서 유형과 다중 문서 유형 모두 가능하다. 특히 다중 문서 유형의 링크를 명백한 링크(Explicit Link)로 구분한다. 후처리가 필요한 질의(Query Requiring Post-processing) 유형에는 정렬 연산과 집단 연산을 사용한 질의로 구분할 수 있다. 정렬 연산 질의는 결과를 알파벳순 또는 오름차순 또는 내림차순으로 정렬한 결과를 만들 수 있다. 집단 연산은 여러 개의 데이터 값을 하나의 스칼라 값으로 나타내는 연산이다. 그러므로, 집단 연산을 사용한 질의는 결과 값이 중복 없이 최대 값, 최소 값, 빈도수, 평균 값으로 의미 있는 데이터를 얻을 수 있다. 표 1은 XML-QL의 질의 특성을 사용해서 데이터를 질의하여 추출, 변환 가능하게 분류하였다.

2.1 XML-QL 예제

2장에서 소개한 질의 유형의 구체적인 예를 들어보자. 예를 들기 위해 XML 문서를 선택해야 한다. DTD를 따르고 영화·음악 정보를 포함하는 XML 문서를 선택한다. 먼저 영화 XML 문서의 DTD[3,9,10]를 살펴보자. MovieList.xml은 MovieList란 루트 엘리먼트를 시작으로 MovieList.dtd를 공통의 DTD로 따르는 한 개 이상의 Movie 엘리먼트로 구성된다. Movie 엘리먼트는 영화제목과 감독, 장르, 출연배우, 영화음악, 제작

년도를 서브엘리먼트로 하고 있다. 또한 영화 음악은 악곡명, 가수, 작곡 년도를 서브엘리먼트로 포함하고 있다. 그리고 영화 감독은 Create_Year라는 link 애트리뷰트를 영화음악은 Mlink 애트리뷰트를 포함하고 있다.

```
<?xml version="1.0"?>
<!ELEMENT MovieList (Movie)+>
<!ELEMENT Movie (Title,Directed_By,Genres,Cast,Music)>
<!ATTLIST Movie year CDATA #REQUIRED>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Directed_By (Director,link)>
<!ELEMENT Director (#PCDATA)>
<!ELEMENT link EMPTY>
<!ATTLIST link CreateYear CDATA #REQUIRED>
<!ELEMENT Genres (Genre)+>
<!ELEMENT Genre (#PCDATA)>
<!ELEMENT Cast (Actor)+>
<!ELEMENT Actor (Firstname,Lastname)>
<!ELEMENT Firstname (#PCDATA)> <!ELEMENT Lastname (#PCDATA)>
<!ELEMENT Music (Mtitle,Singer,Mlink)>
<!ELEMENT Mtitle (#PCDATA)>
<!ELEMENT Singer (fn,ln)>
<!ELEMENT fn (#PCDATA)> <!ELEMENT ln (#PCDATA)>
<!ELEMENT Mlink EMPTY>
<!ATTLIST Mlink MusicYear CDATA #REQUIRED>
```

MusicList.dtd는 노래 제목과 제작자, 가수, 장르와 같은 음악 정보를 엘리먼트로 하고 Music id를 애트리뷰트로 한다.

```
<?xml version="1.0"?>
<!ELEMENT MusicList (Music)+>
<!ELEMENT Music (Title,Produced_by,Singer,Genre)>
<!ATTLIST Music id CDATA #REQUIRED>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Produced_by (#PCDATA)>
<!ELEMENT Singer (Firstname,Lastname)>
<!ELEMENT Genre (#PCDATA)>
<!ELEMENT Firstname (#PCDATA)>
<!ELEMENT Lastname (#PCDATA)>
```

위 DTD 구조를 따르는 MusicList XML 문서는 다음과 같다.

```
<MusicList>
  <Music id="1997">
    <Title>Vison of Love</Title> <Produced_by>
    Columba </Produced_by>
    <Singer><Firstname>Mariah</Firstname>
    <Lastname>Carey</Lastname></Singer>
    <Genre>Popsong</Genre>
  </Music>
  <Music id="1993">
    <Title>Emotions</Title>
    <Produced_by>Columba</Produced_by>
    <Singer>
      <Firstname>Mariah</Firstname>
      <Lastname>Carey</Lastname>
    </Singer>
    <Genre>Popsong</Genre>
  </Music> ...
</MusicList>
```

위 XML 문서는 MusicList를 루트 엘리먼트로 하고 한 개 이상의 Music 엘리먼트를 포함하고 있다. Music 엘리먼트는 제목, 제작자, 가수, 장르로 구성되는 서브엘리먼트를 포함하고 Music 엘리먼트는 id라는 애트리뷰트를 가지며 Singer라는 서브엘리먼트도 포함한다.

또한 영화 정보를 포함하는 MovieList XML 문서는 MovieList.dtd를 따르며 MovieList를 루트 엘리먼트로 하고 여러 개의 Movie 엘리먼트를 서브엘리먼트로 포함한다. 제목, 제작자, 제작 년도, 장르, 배우, 주제곡, 주제곡을 부른 가수, 제작 년도를 정보로 포함한다.

```
<MovieList>
  <Movie year="1977">
    <Title>Star Wars</Title>
    <Directed_By>
      <Director>George Lucas</Director><linkCreateYear=
      "1980"/> </Directed_By>
    <Genres>
      <Genre>Adventure</Genre><Genre>Action</Genre>
      <Genre>Sci-Fi</Genre>
    </Genres>
    <Cast>
      <Actor><Firstname>Mark</Firstname><Lastname>Hamil
      </Lastname> </Actor> ...
    </Cast>
    <Music>
      <Mtitle>Oditsay</Mtitle><Singer><fn></fn><ln>
      </ln></Singer>
      <Mlink MusicYear="" />
    </Music>
  </Movie> ...
</MovieList>
```

위와 같은 XML 문서를 대상으로 표 1에서 소개한 8 가지 질의를 할 수 있다[9]. 그러나 여기서는 지면 제약 상 8가지 유형 중 대표적인 세 가지 질의에 대해 구체적인 예를 설명하겠다.

2.1.1 Query Requiring Non-post-processing/Predicate Based Query의 예)

MovieList.xml과 MusicList.xml의 노래 제목이 같을 때, 제목을 찾으시오.

이 예는 두 개의 XML 문서에서 프레디كت 족, 노래 제목을 비교해서 같을 경우, 그 때의 노래 제목을 찾는 질의이고 XML-QL로 표현하면 다음과 같다.

```
Construct <Result>{
  Where
    <MovieList>
      <Movie> <Music> <Mtitle>SM1</Mtitle>
      <Music> </Movie>
    </MovieList> in "MovieList.xml",
    <MusicList>
      <Music> <Title>St</Title> </Music>
    </MusicList> in "MusicList.xml", text(SM1) =
    text(St)
  construct <Mtitle>SM1</MTitle>
} </Result>
```

2.1.2 Query Requiring Non-post processing/Traversal Based Query의 예)

MovieList.xml에서 MovieList의 하위 엘리먼트가 몇 개가 되든 상관없이 Lastname이나 ln을 모두 찾으시오.

```
Construct <Result>
  Where
    <MovieList>
      <*.Lastname|ln>$ln</>
    </MovieList> in "MovieList.xml",
    Construct <Lastname>$ln</Lastname>
  } </Result>
```

2.1.3 Link based Query/ Explicit Link의 예)

MovieList.xml의 MusicYear가 MusicList.xml을 참조하고 있을 때, 작곡 년도와 그 때 노래 제목을 찾으시오. (단, 작곡년도는 MusicYear이고 노래 제목은 Title이다.)

```
Construct <Result>
  Where
    <MovieList>
      <Movie>
        <Music> <Mlink MusicYear=$y/>
        </Music>
      </Movie>
    </MovieList> in "MovieList.xml",
    <MusicList>
      <Music>
        <Title>$t</Title>
        <Music MusicYear=$my/>
      </Music>
    </MusicList> in "MusicList.xml", text($y) = text($my)
    Construct <Music>
      <MusicYear>$y</MusicYear>
      <Title>$t</Title>
    </Music>
  } </Result>
```

위에서 표현한 XML-QL 질의를 관계형 질의 언어로 변환하기 위해 두 가지 질의 언어의 특징을 비교할 필요가 있다. XML-QL의 질의 특징 중 SQL과 차이점은 몇 가지가 있다. 첫째, XML-QL 결과가 XML 문서로 생성되는 반면에 SQL은 투플 집합으로 생성된다. 둘째, XML-QL은 결과에 엘리먼트 이름과 함께 그 값을 결과로 생성하지만 SQL 결과는 그렇지 못하다. 셋째, XML-QL은 정규 경로 표현을 제공하고 있지만 SQL은 제공하고 있지 않다. 넷째, XML-QL은 링크로 연결하여 결과를 생성할 수 있지만 SQL은 제공하고 있지 않다. 다섯째, XML-QL은 업데이트 질의를 제공하지 않지만 SQL은 업데이트 질의를 제공하고 있다. 이러한 차이점이 있지만 XML 데이터를 효과적으로 검색해서 원하는 데이터를 얻기 위해 프로그래밍을 통해 XML-QL을 SQL로 매핑을 시켜야 한다. 또한 SQL

로 변환할 때, XML-QL에서 제공하고 있지 않은 업데이트 질의를 사용할 수 있다. 따라서 XML-QL을 SQL로 변환시키는 것은 의미가 있으며 그 알고리즘을 4장에서 설명한다.

3. XML 문서 저장

XML-QL 질의를 SQL로 변환하기 위해서 먼저 XML 문서를 관계형 데이터베이스 시스템에 저장해야 한다. 저장한 방식에 따라 질의 방법이 달라지므로 저장 방법에 대한 연구가 필요하다. 많은 관련연구[11,4,5]가 있지만 3장에서는 기존 연구와 다른 저장 방법을 설명한다. 본 연구에서는 스키마 매핑에 [4]의 shared inlining 방법을 적용하였고 여기에 정규 경로 표현 질의와 XML 문서에 삽입과 삭제, 업데이트를 처리하도록 DFS(Depth First Search) Numbering을 더하였다. 또한 한 개의 XML 문서가 여러 개의 테이블로 분리되어 저장될 경우 이러한 테이블 사이에 관계정보를 나타내주는 IndexNReference 테이블 생성을 제안한다. XML 문서를 저장하는 알고리즘을 자세히 살펴보면 다음과 같이 세 단계로 구성된다. 첫째, XML DTD를 사용하여 스키마를 매핑한다. 즉, XML 문서의 구조 정보를 포함하는 DTD를 이용하여 DTD 그래프를 만들고 관계형 데이터베이스 시스템의 스키마로 매핑한다. 이 때, DTD 그래프에서 반복되는 엘리먼트와 다중입력 애지를 갖는 엘리먼트는 다른 테이블에 분리하여 저장한다. 그러므로 엘리먼트가 중복 저장되는 경우를 없앤다. 스키마 매핑 방법은 DTD 그래프에서 알 수 있는 엘리먼트와 애트리뷰트 뿐만 아니라 그 엘리먼트와 애트리뷰트의 id와 부모 엘리먼트 id와 isroot의 정보, 자식 엘리먼트의 수와 그래프에서의 엘리먼트의 깊이 정보를 스키마로 매핑한다. isroot는 서브엘리먼트를 갖고 있는지 아니면 값을 가지고 있는지 isroot의 yes와 no 값으로 알 수 있다. 데이터 유형은 엘리먼트 id와 부모 id와 isroot는 varchar형 그리고 자식 수와 깊이는 int형으로 매핑한다. 엘리먼트 id를 불일 때, DFS Numbering 방식을 사용한다. 노드의 깊이마다 2자리수를 더한다. 즉, 루트 노드를 0으로 하고 그 자식 노드가 5개 있다면, 000, 001, 002, 003, 004로 DFS Numbering을 한다. 이렇게 하면 id와 깊이 정보와 isroot 정보를 사용해 정규 경로 표현이 가능하게 된다. 둘째, 스키마 매핑을 기반으로 XML 문서에 포함된 데이터를 관계형 데이터베이스 시스템의 내용으로 저장한다. 셋째, 테이블 사이의 관계를 나타내는 IndexNReference 테이블을 만든다. 즉, IndexNReference 테이블은 한 개의 XML 문서가

여러 개의 테이블로 저장될 경우에 몇 개의 테이블에 나눠 저장되는지, 질의가 들어왔을 때, 조인해야 하는 테이블 사이의 기본 키와 외래 키에 대한 정보, 링크가 발생될 때 어느 테이블에서 링크가 발생하는지에 대한 정보를 포함하고 있다. 2.1의 영화에 대한 XML 문서는 다음과 같이 저장된다. MovieList.xml의 DTD를 DTD 그래프로 표현하면 그림 1과 같다.

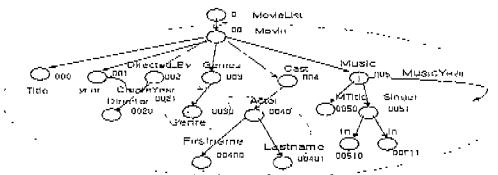


그림 1 MovieList.xml의 DTD 그래프

이 DTD 그래프를 사용해 스키마 매핑을 하려면 앞에서 설명한 알고리즘에 따라 매핑한다. 즉, 한 개 이상의 엘리먼트 노드로 구성된 엘리먼트는 다른 테이블에 분리하여 저장하도록 스키마 매핑을 한다는 규칙에 따라 MovieList.xml은 Movie, Genre, Actor 테이블에 분리해 저장한다. 이와 동시에 엘리먼트를 저장할 때, DFS Numbering 순서로 저장한다. 엘리먼트 뿐만 아니라 엘리먼트 노드 id와 부모 엘리먼트 노드 id와 서브엘리먼트 노드를 포함하는지 확인할 수 있는 자식의 수와 DTD 그래프에서 그 노드의 깊이 정보도 함께 저장한다. 그리고 엘리먼트 id와 부모 id와 isroot의 데이터 유형을 varchar형으로 매핑하고 자식 수와 깊이의 데이터 유형은 int형으로 한다. Movie 테이블 스키마는 표 2와 같다. Movie 테이블과 같이 Actor와 Genre 테이블도 스키마 매핑을 할 수 있다. Movie 테이블이 주요 테이블이고 Actor와 Genre 테이블을 조인하므로 MovieList.xml에 있는 데이터를 추출해 낼 수 있다. 따라서 Movie 테이블의 Movieid가 기본 키이고 Actorid와 Genreid가 외래 키로 선언된다. Actor와 Genre 테이블도 Movie 테이블과 같은 스키마를 사용하여 저장하였다. MusicList.xml 문서도 MovieList.xml 문서처럼 스키마 매핑을 할 수 있다. Musiclist XML 문서는 한 개의 테이블로 매핑된다. 앞에서 소개한 MovieList와 Musiclist XML 문서는 서로 연관성이 있는 문서이다. 즉, MovieList의 MusicYear은 MusicList XML 문서의 MusicYear 엘리먼트를 link하고 있다. 서로 관련 있는 문서 사이의 연결 정보가 필요하다. 또한 몇 개의 테이블로 나누어져 있는지도 중요하다.

표 2 Movie 테이블의 스키마

```

List(Listid: varchar, Listisroot: varchar, ListChild_num: integer, Listdepth: integer)
Movie(Movicid: varchar, primary key, Movieparentid: varchar, Movieisroot: varchar, MovieChild_num: integer, Moviedepth: integer)
Title(Titleid: varchar, Titleparentid: varchar, Titleisroot: varchar, TitleChild_num: integer, Titleddepth: integer, Title: varchar)
Genres(Genresid: varchar, foreign key)
Actor(Actorid: varchar, foreign key)
Music(Musicid: varchar, Musicparentid: varchar, Musicisroot: varchar, MusicChild_num: integer, Musicdepth: integer)
Mtitle(Mtitleid: varchar, Mtitleparentid: varchar, Mtitleisroot: varchar, Mtitleddepth: integer, Mtitle: varchar)
Msinger(Msingerid: varchar, Msingerparentid: varchar, Msingersroot: varchar)

```

따라서 이러한 XML 문서와 테이블 사이의 정보를 포함하는 IndexNReference 테이블을 생성한다. 이 테이블은 DTDid, DTD_name, XML_Doc_name, Table_num, Table_name, Primary_key, Foreign_key, Internal_link_target_table, Internal_link, External_link_target_table, External_Link, External_link_foreign_key 정보를 포함하고 있다. IndexNReference 테이블을 사용해서 XML-QL 질의를 SQL 질의로 변환을 더 효과적으로 할 수 있다. 4장에서 XML-QL 질의를 SQL로 변환시키는 번역기 알고리즘을 설명한다.

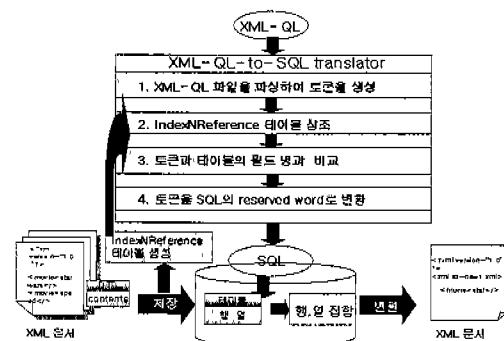


그림 2 XML-QL to SQL 번역기 설계도

4. XML-QL to SQL 번역기 설계

4.1 XML-QL을 SQL로 변환시키는 번역기 알고리즘

3장에서 XML 문서를 저장하고 여러 개의 테이블 사이의 관계정보를 포함하는 IndexNReference 테이블을

만들었다. 이 테이블 정보를 사용해 XML-QL 질의를 SQL로 변환시키는 번역기 알고리즘은 다음과 같다. 먼저, XML-QL 질의를 받아들였을 때, 이것을 파싱하여 토큰 단위로 배열에 저장해 두고 다음 알고리즘을 적용하면 된다.

```

if( token == *.xml ) {
    IndexNReference();
    print(from 테이블 이름);
} //①
if( token == 조건 )
    print(where 조건); //②
if( token == ?, 조건) {
    조건 : 변수x와 y가 같을 때,
    if(token == $x) //토큰이 변수와 같을 때, 변수 x의
        엘리먼트(xc)가 테이블 A에 있고
        if (token == $y)
            //변수 y의 엘리먼트(ye)가 테이블 B에 있다면,
            //테이블에서 변수에 해당하는 엘리먼트와 필드
            이름을 비교한다.
            search(table_field);
        if(table_fieldname == xe) //테이블 A의 필드
            이름이 xe와 일치하고
        if(table_fieldname == ye) //테이블 B의 필드
            이름이 ye와 일치할 때,
            print(from A inner join B on A.x = B.y);
} //③
if( token == construct ) {
    if(where에 있는 변수의 엘리먼트 = construct
       에 있는 변수의 엘리먼트)
        if (변수의 엘리먼트 = 테이블 필드 이름)
            print(select 테이블의 필드 이름);
} //④

```

① 토큰이 where 절에서 *.xml일 때, IndexNReference 테이블을 먼저 참조해서 한 개의 XML 문서가 몇 개의 테이블로 분리되어 저장되는지 확인한다. 한 개의 테이블에 저장되고 *.xml 다음에 커머(,)가 없다면 즉, 조건이 없다면 SQL의 예약어 from과 함께 해당 테이블 이름을 출력한다. ② 한 개의 테이블에 저장되고 조건이 있다면, SQL의 where 절에 조건을 출력한다. ③ 토큰이 where 절에서 *.xml일 때, IndexNReference 테이블을 참조한 결과 2개 이상의 테이블에 분리 저장되고 조건이 있다면 SQL 예약어 from을 출력하기 전에 IndexNReference 테이블을 참조해서 조건에 나타난 변수가 있는 엘리먼트를 테이블의 필드 이름과 비교해서 from A join B on 형태로 출력한다. ④ construct 절에서 엘리먼트 다음에 변수가 있을 때, 변수에 해당하는 엘리먼트를 기억해 두고 이 변수와 where 절의 변수가 같은지 비교한다. 일치할 경우, 그 때의 엘리먼트를 기억해 두고 IndexNReference 테이블을 통해 얻은 테이블 정보를 사용해서 해당 테이블에서 엘리먼트와 테이블의 필드 이름을 비교한다. 같으면 select 테이블 이름.필드 이름을 출력한다. 위와 같은 알고리즘으로 XML-QL 질의가 SQL로 변환된다. 특별히, 한 개의 XML 문서에서 링크를 기본 질의를 할 때와 두 개

의 XML 문서에서 링크를 사용해서 데이터를 검색할 때는 XML 문서가 한 개 또는 2개 이상의 테이블로 저장된다. 그러므로 데이터를 추출하기 위해 테이블을 조인해야 한다. 그런데 이런 질의 유형과 몇 개의 테이블로 저장되느냐에 따라 조인할 때, where 절이 달라진다. 1개의 XML 문서가 1개의 테이블로 저장이 될 때는 별로 문제가 되지 않지만 1개 또는 2개의 XML 문서가 2개 또는 그 이상의 테이블로 저장이 될 때는 XML-QL 질의 유형이 SQL로의 변환이 달라진다. 특히 조건을 나타내는 where 절에서 달라지는 데 질의 유형 Q1, Q3, Q6, Q7, Q8은 where A 테이블 inner join B 테이블 on 형식으로 변환되고 후처리가 필요한 연산(sort, aggregation)은 where A 테이블 right(또는 left) outer join B 테이블 on 로 변환된다[9]. A, B 테이블은 데이터를 검색하기 위해 조인해야 하는 테이블이다.

4.2 결과를 XML 문서로 변환하는 알고리즘

4.1에서 설명한 XML-QL to SQL 번역기 알고리즘을 실행하면 튜플 집합이 결과로 나온다. 이 튜플 집합을 애플리케이션이나 웹에서 재사용 가능하도록 XML 문서로 변환시켜 주는 작업도 중요하다. Tuple to XML 변환 알고리즘은 다음과 같다.

- 1 SQL 결과를 파일로 저장한다.
- 2 이 파일을 읽어 들인다.
- 3 <? XML version "1.0"?>을 출력 파일에 삽입한다.
- 4 <XML ID=newQnum.xml>을 삽입한다.(Qnum은 Query number임)
- 5 <Result>를 삽입한다.
- 6 SQL 결과의 필드 이름을 태그 엘리먼트로 사용한다.
- 7 태그를 연다.
- 8 튜플 셋을 content로 만든다.
- 9 태그를 닫는다.
- 10 <Result>를 삽입한다.

5. XML-QL to SQL 번역기 구현

4장에서 설명한 번역기와 변환기 알고리즘을 토대로 번역기를 구현하였다. 2장에서 분류한 8가지 유형 중 2.1.1 예를 번역기에 실행시키면 다음과 같은 SQL로 변환된다.

```

select Movie.Mtitle
from Movie inner join Music on Movie.Mtitle =
Music.Tmttitle

```

그림 3 Query requiring Non-Post-processing/ Predicate Based Query의 XML-QL을 SQL로 번역한 결과

위의 SQL을 query analyzer를 사용해 실행하면 그림 4와 같다.

```
Mtitle
```

```
Vision of Love
My Heart will go on
My Heart will go on
(3 row(s) affected)
```

그림 4 XML-QL을 SQL로 번역하여 실행한 화면

이렇게 만들어진 템플릿집합을 XML 문서로 변환하기 위해 <?xml version="1.0"?> <XML ID= new6.xml> 을 XML 문서의 헤더로 삽입하고 <Result>라는 엘리먼트를 삽입해서 XML 문서의 루트 엘리먼트로 만든다. 그 다음 템플릿집합의 애트리뷰트 Mtitle을 엘리먼트 태그로 만든다. Mtitle의 실제 값은 엘리먼트로 만든다. 엘리먼트의 수는 템플릿 수만큼 만들어진다. 그 다음 </Result>를 삽입하여 XML 문서 형식을 마무리한다. 이러한 고리즘에 따라 그림 5와 같은 XML 문서로 변환된다. 이 결과는 애플리케이션에서 곧바로 재사용할 수 있다.

```
<?xml version="1.0"?>
<XML ID=new6.xml>
<Result>
    <Mtitle> Vision of Love </Mtitle>
    <Mtitle> My Heart will go on </Mtitle>
    <Mtitle> My Heart will go on </Mtitle>
</Result>
```

그림 5 SQL 실행 결과를 XML 문서로 변환한 화면

2.1.2에서 Lastname 또는 ln을 찾는 Query Requiring Non-post-processing/ Traversal Based Query 유형의 XML-QL은 다음과 같은 SQL질의로 변환된다.

```
select Movie.ln
from Movie
union
select Actor.Lastname
from Actor inner join Movie on Actor.Actorparentid =
Movie.castid
```

그림 6 Query Requiring Non-post-processing/Traversal Based Query의 XML-QL을 SQL로 번역한 결과

```
ln
-----
Baker(I)
Caesar
Colley
Daniels
Dee Williams
Doyokaya
Dukaprio
Earl Jones
Fisher
Hamil
...
Carey
Dion
(30 row(s) affected)
```

그림 7 Query Requiring Non-post-processing/Traversal Based Query의 XML-QL을 SQL로 번역하여 실행한 결과

```
<?xml version="1.0"?>
<XML ID=new2.xml>
<Result>
    <ln> Baker(I) </ln>
    <ln> Caesar </ln>
    <ln> Colley </ln>
    <ln> Daniels </ln>
    <ln> Dee Williams </ln>
    <ln> Doyokaya </ln>
    <ln> Dukaprio </ln>
    ...
    <ln> Carey </ln>
    <ln> Dion </ln>
</Result>
```

그림 8 SQL 실행 결과를 XML 문서로 변환한 화면

2.1.3에서 노래 제목과 작곡 년도를 찾는 Link based Query/ Explicit Link 유형의 XML-QL은 다음과 같은 SQL질의로 변환된다.

```
select Music.Tmttitle, Music.MusicYear
from Music left outer join Movie on Music.Music
Yearparentid = Movie.MusicYearlink
```

그림 9 Link based Query/ Explicit Link의 XML-QL을 SQL로 번역한 결과

Tmttitle	MusicYear
Vision of Love	1997
Emotion	1993
Mai_piui_cosi_lontano_rns	1998
Vision	1999
Because You Love me	1994
Summer_nights	1996
My Heart will go on	1984

그림 10 Link based Query/Explicit Link의 XML-QL을 SQL로 번역하여 실행한 결과

```

<?xml version="1.0"?>
<XML ID=new8.xml>
<Result>
<MusicList>
    <Title>Vision of Love </Title> <MusicYear> 1997
    </MusicYear>
    <Title>Emotion </Title> <MusicYear>1993
    </MusicYear>
    <Title> Mai_piu_cosi_lontano_rns </Title>
    <MusicYear> 1998</MusicYear>
    <Title>Vision </Title>
    <MusicYear>1999</MusicYear>
    <Title>Because You Love me </Title>
    <MusicYear>1994 </MusicYear>
    <Title>Summer_nights </Title>
    <MusicYear>1996 </MusicYear>
    <Title>My Heart will go on </Title>
    <MusicYear>1981 </MusicYear>
<MusicList>
</Result>

```

그림 11 SQL 실행 결과를 XML 문서로 변환한 화면

6. 결 론

지금까지 관계형 데이터베이스 시스템에 XML을 저장하고 저장된 XML 데이터를 검색하는 방법을 설명하였다. 먼저, XML-QL의 질의 유형을 분류하였고 XML-QL 질의 언어가 들어왔을 때, SQL 질의로 번역시키는 XML-QL to SQL 번역기를 구현하였다. 또한 질의 결과를 XML 문서로 변환해 애플리케이션에서 사용할 수 있도록 하였다. 본 논문의 의의는 번역기의 가능성을 보여주고 있다. 그리고 XML 질의의 포괄적인 분류와 그에 대한 SQL 관계형 질의로의 매핑을 보여주고 있다. RDBMS와 XML 전용 서버를 사용한 번역기 기반 접근법의 차세한 성능 비교는 본 논문의 범위에 해당되지 않는다. 향후 연구 과제로 현재 XML-QL이 제공하지 않고 있는 집단연산 함수와 업데이트 질의가 보강된다면 더 완벽한 XML-QL이 될 것이다. 이런 질의가 보강될 때, SQL로 번역하는 추가적인 번역기 설계가 이뤄지면 효과적일 것이다. 또한 XML 문서를 자동 저장하는 연구가 이뤄져야 할 것이다.

참 고 문 헌

- [1] Serge Abiteboul, Peter Buneman, Dan Suciu, *Data on the web from relations to semistructured Data and XML*, Morgan Kaufmann, 1999.
- [2] Peter Buneman, Susan Davidson, Mary Fernandez, Dan Suciu, Adding structure to unstructured data, University of Pennsylvania, Computer and Information Science Department Technical Report, Number MS-CIS-96-21, 1996.
- [3] Extensible Markup Language (XML) 1.0 (Second Edition), .
- [4] Jayavcl Shanmugasundaram, Kristin Tufte, Gang He, Relational Databases for Querying XML Documents:Limitationas and Opportunities,In *Proc. conf. on VLDB* 1999.
- [5] Daniela Florescu, Donald Kossmann, "A Performance Evaluation of Alternative Mapping Schemas for Storing XML Data in a Relational Database," <http://www.inria.fr/RRRT/publications-eng.html>.
- [6] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, Janet L.Wiener, The Lorel Query Language for Semistructured Data, *International Jornal on Digital Libraries*, 1(1) :68-88, April 1997.<http://www-db.stanford.edu/lore/pubs/data.html>.
- [7] Alin Deutsch, Mary Fernandez, Dan Suciu, "Storing Semistructured Data with STORED," *SIGMOD*, 1999, <http://www.www8.org/fullpaper.html>.
- [8] A. Deutsch, M. Fernandez, D. Florescu, Alon Levy and D.Suciu. XML-QL : A Query Language for XML, In *Proc. Of the Query Languages workshop (QL98)*, Cambridge, Mass., December 1998,<http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/>.
- [9] 장경자, XML 질의 언어를 SQL로 변환시키는 번역기, 이화여자대학교, 1999.
- [10] William J.Pardi, *XML in Action*, Microsoft Press, 1999, <http://www.microsoft.com/xml>.
- [11] Alin Deutsch, Mary Fernandez, Dan Suciu, Storing Semistructured Data with STORED, *SIGMOD*, 1999, <http://www.www8.org/fullpaper.html>.



장 경 자

1998년 한림대학교 컴퓨터공학과(학사). 2000년 이화여자대학교 컴퓨터학과(석사). 2000년 ~ 현재 삼성종합기술원, 멀티미디어랩, MPEG-4 AFX 국제표준화팀 연구원. 관심분야는 XML 및 웹, 멀티미디어 데이터 처리, MPEG-4 AFX, 3D 그래픽스



이 기 호

1961년 이화여자대학교 수학과 학사, 석사. 1972년 텍사스 대학교 전산학 석사, 박사 수료. 1981년 서울대학교 전산학 박사. 캘리포니아대 연구교수, 이화여대 공대학장, 대학원장 역임. 1973년 ~ 현재 이화여대 컴퓨터학과 교수. 관심분야는 인터넷언어, XML기반 전자상거래