XQuery SQL:2003 번역기 설계 및 구현

(Design and Implementation of a Translator form XQuery to SQL : 2003)

김 송 현 † 박 영 섭 †† 이 윤 준 †††
(Song Hyon Kim) (Young Sup Park) (Yoon Joon Lee)

요 약 XML은 다양한 장점으로 인해 인터넷 기반 환경에서 데이타 표현 및 교환의 표준으로 자리잡았다. XML이 데이타 표현 및 교환의 주요한 포맷으로 자리잡으면서 XML 데이타의 효율적인 저장 및 질의 처리에 대한 연구가 활발히 진행되었다. XML 데이타를 관계형 데이타 베이스시스템에 저장하는 것은 데이타 관리와 질의 처리에서 많은 이익을 가져온다. 왜냐하면, 관계형 데이타 베이스 시스템은 강력한 질의 처리 및 데이타 관리 기능을 제공하고, 이들 기능을 확장하여 XML 데이타에 적용할 수 있기 때문이다. 그러나, 이 방법을 사용하기 위해서는 XML 질의를 SQL 질의로 변환해야 한다. 본 논문에서는 대표적인 XML 질의 언어인 XQuery 질의를 SQL:2003 질의로 변환하는 질의 번역기를 설계 및 구현한다. SQL:2003은 SQL:1999을 대체하는 최신 SQL 표준으로, XML을 지원하기 위한 SQL/XML을 정의하고 있다. 본 논문의 주요 공헌은 다음과 같다. 첫째, SQL:2003 표준에서 정의하고 있는 XML 지원 특징을 살펴보고 미흡한 부분에 대한 사용자 정의 함수를 제안한다. 둘째, XQuery 질의를 SQL:2003 표준을 준수하는 SQL 질의로 변환하는 방법을 제안한다. 셋째, 번역기에 대한 설계와 구현을 자세히 기술하여 번역기의 가능성(feasibility)을 보여준다.

키워드 : XQuery, SQL, 질의 변환, SQL/XML, SQL:2003

Abstract Due to its diverse advantages, XML has secured its position as a standard for data representation and exchange in the Internet. As a consequence, there has been much research on efficient storing and query processing of XML data. Storing XML data in a relational database system warrants much benefit in data management and query processing; the system provides a strong query processing and data management function and can be applicable to XML data, its function being extended. In this paper, we design and implement a query translator that translates XQuery, a representative XML query language, into SQL:2003 query. SQL:2003, the latest SQL standard used as a substitute for SQL:1999, defines SQL/XML that supports XML. The main contribution of this paper is as follows: First, we look into the supporting features of XML, defined in the SQL:2003 standard, and propose a user-defined function for shortcoming sections. Second, we propose a way to translate XQuery into SQL that observes the latest SQL standard. Third, we describe in detail the design and the implementation of the translator to show its feasibility as a translator.

Key words : XQuery, SQL, Query translation, SQL/XML, SQL:2003

1. 서 론

XML(eXtensible Markup Language)[1]은 기존에 사용하던 HTML의 한계를 극복하고 SGML의 복잡함을 해결하는 방안으로 1998년 W3C(World Wide Web Consortium)에 의해 발표되었다. XML은 다양한 장점으로 인해 인터넷 기반 환경에서 데이타의 표현 및 교환을 위한 표준으로 자리잡았다. XML이 데이타 표현 및 교환의 표준으로 자리 잡으면서 많은 새로운 데이타들이 XML 형식으로 작성되고, 기존의 데이타들이 XML 형식으로 변환되어, XML 데이타의 양이 크게 증가하고 있다. 따라서 대량의 XML 데이타에 대한 효율적인 저장 및 질의 처리가 매우 중요하게 인식되고 있으며, 최근 몇 년 동안 XML 데이타의 저장 및

† 정 회 원 : 공군사관학교 전산통계학과
shkim@afa.ac.kr
†† 정 회 원 : 한국과학기술원 전산학과
yspark@dbserver.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
yjlee@dbserver.kaist.ac.kr
논문접수 : 2005년 11월 28일
심사완료 : 2006년 8월 22일

질의 처리 기법에 대한 연구가 활발하게 진행되고 있다.

XML 데이타의 저장 및 질의 처리 기법 중에서 기존의 데이타베이스 시스템을 기반으로 한 XML 저장 및 질의 처리 시스템에 관한 연구가 활발히 진행되고 있는데, 이는 기존 시스템들이 오랜 기간 동안의 연구, 개발 및 운영을 통해 안정화되고 최적화된 성능을 가진 다양한 기능을 제공하고 있기 때문이다. 기존의 많은 데이타들이 관계형 데이타베이스 시스템에 저장되어 있고, 앞으로도 관계형 데이타베이스 시스템의 사용은 지속될 것으로 예상되기 때문에 관계형 데이타베이스 시스템을 기반으로 한 XML 데이타 저장 및 질의 처리 시스템은 기존의 데이타들과의 통합 처리를 효율적으로 지원할 수 있는 등의 장점을 갖추고 있어 그 연구가 더욱 활발히 진행될 것으로 예상된다.

XML 데이타를 관계형 데이타베이스 시스템에 저장하고 질의를 처리하고자 할 때, 크게 두 가지 관점에서 이슈가 발생한다. 첫 번째는 XML 데이타를 효율적으로 저장하기 위하여 관계 스키마(Relational schema)를 어떻게 생성할 것인가 하는 것이고, 두 번째는 관계형 데이타베이스 시스템에서 XML 질의를 처리하기 위하여 XML 질의를 어떻게 효율적으로 SQL 질의로 변환할 것인가 하는 것이다.

전통적인 관계형 데이타베이스 시스템은 주로 SQL:1999[2]를 준수하고 있는데, SQL:1999는 XML을 지원하기 위한 표준을 정의하고 있지 않다. 따라서 전통적인 관계형 데이타베이스 시스템을 기반으로 하는 XML 저장 및 질의 처리 시스템 연구에서는 독자적인 관계 스키마를 생성하여 XML 데이타를 저장하고, 관계 스키마에 종속적으로 XML 질의를 SQL로 변환하는 방법을 제안하고 있다. 즉, XML 데이타 저장 방법과 XML 질의의 SQL 변환 방법이 종속적인 관계에 놓이게 된다.

최근 XML과 관계형 데이타베이스의 상호운용에 대한 요구가 증대되면서 SQL:1999 표준을 대체하는 SQL:2003[3]에는 XML을 지원하기 위한 표준을 포함하고 있다. 그러므로, SQL:2003을 지원하는 관계형 데이타베이스 시스템을 기반으로 한다면, XML 질의를 보다 쉽게 SQL 질의로 변환 할 수 있으며, 관계형 데이타베이스 시스템의 결과를 XML 형태로 구조화하고 태깅(tagging)을 하던 후처리 과정을 생략하고 관계형 데이타베이스 시스템에서 바로 XML 인스턴스를 SQL 결과로 얻을 수 있는 장점이 있다.

SQL:2003은 SQL:1999에 있던 버그들을 수정하고, SQL:1999의 내용 중 중복되거나 불필요한 부분을 제외하고 새롭게 8개의 부분(part)으로 정비하였으며, XML과 관련된 SQL/XML(part 14)을 추가하였다.

SQL/XML은 네이티브(native) XML 데이타 타입, SQL과 XML의 매핑 규칙, XML 출판(publishing) 함수 등을 정의한다. XML에 대한 네이티브 데이타 타입을 정의함으로써, SQL 질의 언어를 사용하여 XML 인스턴스를 조작하는 것을 가능하게 한다. SQL/XML에는 표준 SQL 데이타 타입을 위한 확장된 전환 규칙(conversion rule) 뿐만 아니라, SQL 식별자를 적법한(qualified) XML 식별자로, 혹은 적법한 XML 식별자를 SQL 식별자로 매핑하는 규칙을 정의한다. XML 출판 함수는 SQL에서 관계 데이타를 이용하여 XML 구조를 생성하는 것을 가능하게 한다. XML 출판 함수들을 정리하면 다음과 같다.

- *XMLELEMENT()* : 태그 이름과 관계 데이타를 전달받아서 XML 엘리먼트를 생성한다.
- *XMLATTRIBUTES()* : 관계 데이타의 각 칼럼 이름을 애트리뷰트의 이름으로 사용하여 여러 칼럼으로부터 XML 애트리뷰트들을 생성한다. 주로, *XMLELEMENT()*와 함께 호출된다.
- *XMLROOT()* : XML 인스턴스의 루트 엘리먼트를 생성한다.
- *XMLFOREST()* : 여러 칼럼의 데이타를 전달받아서 XML 엘리먼트들을 생성한다. 이때, 명시적으로 태그 이름을 전달 받지 않으면, 관계 데이타의 칼럼명이 태그 이름으로 사용된다.
- *XMLCONCAT()* : 관계 데이타 결과에서 한 행의 여러 칼럼에 대한 XML 엘리먼트 리스트를 전달받아서 XML forest를 생성한다.
- *XMLAGG()* : 관계 데이타 결과에서 여러 행의 한 칼럼에 대한 XML 엘리먼트 리스트를 전달받아서 XML forest를 생성한다.

본 논문은 SQL:2003을 준수하는 관계형 데이타베이스 시스템을 기반으로 하여 XML 데이타가 저장되어 있을 때 발생하는 XML 질의의 SQL 변환 문제를 다룬다. 본 논문의 구성은 다음과 같다. 2장에서는 기존 연구에서 XML 질의의 SQL 변환을 살펴보고, 3장에서는 SQL:2003 표준에서 정의하고 있는 XML 지원 특징 중에서 미흡한 부분에 대한 사용자 정의 함수를 제안하고, SQL 템플릿을 기반으로 한 XML 질의의 SQL 변환에 대해 설명한다. 4장에서는 XQuery 질의를 SQL 질의로 변환하는 질의 번역기의 설계와 구현에 대해 설명하고, 5장에서 결론을 맺는다.

2. 관련 연구

그 동안 XML 질의를 SQL로 변환하여 처리하고자

하는 노력이 많이 진행되어 왔다[11-17]. 일반적인 XQuery를 SQL로 변환하고자 하는 연구는 [16], [17]에서 진행되었다.

[16]과 [17]은 에지 매핑[18] 방법으로 XML 데이타를 관계형 데이타베이스 시스템에 저장하고, XQuery 질의에 대한 SQL 변환은 사전에 정의된 SQL 템플릿을 이용하여 단일 SQL로 변환한다. 그러나, XML 데이타를 관계형 데이타베이스에 저장하는 방법을 비교한 논문[19]에서 에지 매핑 방법은 스키마를 기반으로 한 저장 방법과 비교했을 때, 떨어진 성능을 보이고 있으며, [16]과 [17]에서 제안한 방법은 모든 XML 데이타를 하나의 테이블에 저장하고 있으므로, XML 데이타의 양이 많은 실제 환경에서는 적용하기 힘들다.

[16]은 XML 인스턴스에 대해 dynmaic interval encoding을 기반으로 하여 XQuery Core의 서브셋을 SQL view 집합으로 변환하는 방법을 제안하였다. 그러나 변환 방법은 XQuery의 의미적 특성을 유지하는 데 부족하다. for 표현식의 변환에서 backmapping step의 생략은 임의의 중첩된 표현식을 사용할 수 없게 하고, encoding이 시퀀스와 document order를 구별할 수 없으므로 시퀀스를 처리하는데 애로가 있다. 특히, [16]에서는 많은 정수 조건에 의한 조인을 수행하게 되는데, 이는 일반적인 관계형 데이타베이스 시스템을 이용할 시 성능저하를 가져오게 된다[24]. [17]도 tree encoding 방법으로 XML 데이타를 테이블에 저장하므로, [16]과 유사한 단점을 가진다.

본 논문에서는 XML 데이타 타입 뷰를 사용하여 XML 질의를 하는데, SQL:2003에서 XML 데이타 타입을 지원하고 있으므로 표준에 의한 질의 처리를 할 수 있다. 뷰를 기반으로 할 때의 장점은 그 하부의 관계 스키마에 영향을 받지 않는다는 것이다. 관계형 데이타베이스 시스템에서 XML 데이타를 저장하는 여러 가지 방법(LOB 저장, XML shredding 등)을 제공하고 있는데, 질의를 변환하는 방법이 특정한 저장 방법을 전제하지 않아도 된다는 것이다. 즉, 데이타를 저장하는 방법과 질의를 처리하는 방법을 전혀 무관하게 처리할 수 있다는 것이다. [16]과 [17]은 저장 방법과 변환 방법이 긴밀하게 연관되는 단점을 가지고 있다.

3. SQL 템플릿을 기반으로 한 질의 변환

본 장에서는 SQL 템플릿을 기반으로 XQuery 질의를 SQL 질의로 변환하는 방법에 대해 설명한다. 제 3.1절에서는 본 논문에서 다루는 XQuery의 범위에 대해 설명하고, 제 3.2절에서는 XQuery 질의를 처리하기 위하여 XQuery 구문을 핵심 구문과 보조 구문으로 구분하는 것에 대해 설명한다. 제 3.3절에서는 제 3.2절에서 정의한 핵심 구문에 대응되는 SQL 템플릿을 정의하고, 제 3.4절에서는 XQuery 질의를 SQL 템플릿을 이용하여 SQL 질의로 변환하는 과정을 설명한다.

3.1 XQuery 분석

XML 문서에 대한 질의 언어로는 Quilt[4], XQL[5], XML-QL[6], XPath[7], XQuery[8] 등과 같은 다양한 XML 질의 언어들이 등장하였다. 이 중에서 XPath와 XQuery가 가장 널리 사용되고 있는 XML 질의 언어이다.

XPath는 XML 문서의 특정한 부분을 지정하기 위한 언어로 XSLT[9]와 XPointer[10]등에서 사용하기 위해 디자인되었다. XPath는 특정한 부분의 위치를 표현하기 위해서 위치 경로(location path)라는 표현을 사용한다. 이 위치 경로를 통해서 문서 안의 위치를 단계(step)별로 표현할 수 있다. XQuery는 XPath를 포함하는 질의 언어로서 현재 W3C에서 표준화가 진행 중이다. XQuery는 SQL의 select-from-where 표현과 유사한 기능을 가지는 FLWOR 표현식을 제공한다. FLWOR 표현식은 for 절, let 절, where 절, order by 절, 그리고 return 절로 구성되며, 각 절 안에서 또 다른 FLOWR 표현식이 중첩될 수 있는 특징이 있다.

XQuery의 두드러진 특징은 강한 타입 언어(strong typed language)라는 것이다. 즉, XQuery의 표현식, 연산자, 함수 등의 피연산자는 반드시 기대 타입(expected type)를 준수해야 한다. 이를 위반하면 XQuery의 처리 단계에 따라 static error, dynamic error, type error 등의 에러가 발생한다[8].

SQL은 XQuery 언어와 비교할 때, 제한된 데이타 타입을 가지고 있어, XML 데이타를 관계형 데이타베이스 시스템에 저장할 때 XML 데이타의 엘리먼트나 애트리뷰트의 데이타 타입을 관계형 데이타베이스 시스템에서 제공하는 데이타 타입으로 타입변환(type casting)하여 저장한다. 또한, 관계형 데이타베이스 시스템에 저장되어 있는 기존 데이타에서 XML 인스턴스를 생성할 때는 데이타 타입 매핑을 통하여 XML 인스턴스를 생성한다. 따라서, 관계형 데이타베이스 시스템을 기반으로 한 XML 데이타 저장 및 검색 시스템에서 XQuery의 강한 타입 체크는 무의미하다. 그러므로, 본 논문에서는 타입과 관련된 표현식, 함수, 연산자를 제외한 XQuery 질의의 SQL로의 변환 방법을 제안한다.

3.2 XQuery 구문 정의

XQuery 구문은 XQuery 질의 내에서의 역할과 SQL 템플릿의 필요성에 따라, 핵심(Core) 구문과 보조(Assistance) 구문으로 분류된다.

정의 1. 핵심 구문은 XQuery 에서 중추적인 역할을 하며, SQL 템플릿을 필요로 하는 구문이다.

핵심 구문은 XQuery 질의를 SQL로 변환했을 때, XML 데이타 타입 인스턴스를 입력으로 받아 XML 데이타 타입 인스턴스를 반환할 수 있는 구문을 말하며, FLWOR 표현식의 ForClause, LetClause, WhereClause, OrderByClause, ReturnClause, 함수호출의 FunctionCall, 생성자인 DirElemConstructor 등이 그 예이다.

정의 2. 보조 구문은 XQuery에서 보조적인 역할을 하며, 대응하는 데이타 표현식을 가지는 구문이다.

보조 구문은 핵심 구문이 SQL 템플릿으로 치환될 때 필요한 데이타를 제공하며 대응하는 SQL 템플릿을 가지지 않는다. 보조 구문은 핵심 구문을 제외한 모든 구문을 가리킨다. 보조 구문은 그들이 가진 데이타를 저장하기 위한 데이타 구조를 가지며, 이러한 데이타 구조를 통틀어 **데이타 표현식**이라 칭한다.

그림 1은 본 논문에서 다루는 XQuery 구문의 주요 부분을 발췌한 것이다.

```
EnclosedExpr ::= <LBrace> Expr <RBrace>
Expr ::= ( FLWORExpr | ComparisonExpr )
FLWORExpr ::= ( ( ForClause | LetClause ) )+
( WhereClause )?
( OrderByClause )? ReturnClause
ForClause ::= <ForVariable> <VarName> <IN> Expr
LetClause ::= <LetVariable> <VarName> <COLONEQUALS> Expr
WhereClause ::= <Where> Expr
OrderByClause ::= <OrderBy> OrderSpec
OrderSpec ::= ExprSingle OrderModifier
OrderModifier ::= ( <Ascending> | <Descending> ) )?
ReturnClause ::= <Return> Expr
// snip
```

그림 1 XQuery 구문의 일부

3.3 SQL 템플릿

SQL:2003 표준에서 XML을 지원하는 SQL/XML은 네이티브 XML 데이타 타입, SQL과 XML의 매핑 규칙, 그리고 XML 출판 함수에 대해서는 정의하고 있지만, XML 데이타 타입 인스턴스에서 XML 데이타를 추출하는 함수는 정의하고 있지 않다. XML 데이타 타입 인스턴스에 대한 데이타 추출 함수가 없으면, XML 인스턴스에서 단편(fragment)이나 애트리뷰트값을 추출할 수 없으므로, SQL/XML에서 정의하고 있는 XML 데이타 타입을 활용하는데 한계가 있다.

따라서, 관계형 데이타베이스 시스템에서 XML을 원활히 지원하기 위하여 XML 데이타 타입 인스턴스로부터 정보를 추출할 수 있는 수단을 가져야 한다. 즉, 문자 타입이나 숫자 타입과 같은 스칼라 SQL 타입에 맞는 XML 인스턴스 값을 추출하거나, XML 데이타 타입으로 표현되는 XML 단편을 추출할 수 있어야 한다.

SQL:2003을 지원하는 여러 데이타베이스 시스템에서도 XML 데이타 타입 인스턴스에서 데이타를 추출하기 위한 함수를 제공하고 있다. 오라클에서는 XML 단편을 추출하기 위한 extract() 함수를 비롯하여 extractvalue(), existnode() 등과 같은 함수를 제공하고 있으며[21], IBM DB2에서는 extractclob(), extractinteger(), extractvarchar() 등과 같은 함수를 제공하여 XML 데이타 타입 인스턴스에 대한 질의를 지원하고 있다[22].

표 1 XML 데이타 타입 인스턴스에 대한 데이타 추출 함수

	XML 단편 추출	컨텐트나 애트리뷰트 값 추출	튜플 집합 생성
Oracle	Extract	extractvalue	xmlsequence
DB2	extractCLOB	extractInteger extractSmallint extractDouble extractReal extractChar extractVarchar extractDate extractTime extractTimestamp	-

본 논문에서는 위에서 논의된 바에 따라 XML 데이타 타입 인스턴스에서 데이타를 추출하는 사용자 정의 함수(User-Defined Function)를 정의한다. XML 데이타 타입 인스턴스에서 추출하는 데이타는 XML 단편, 컨텐트(content) 그리고 애트리뷰트 값으로 분류할 수 있다. 본 논문에서는 XML 데이타 타입 인스턴스에서 XML 단편을 추출하기 위해 *XMLEXTRACT()* 함수를 정의하고, XML 데이타 타입 인스턴스의 컨텐트나 애트리뷰트 값을 추출하기 위해 *XMLEXTRACTVALUE()* 함수를 정의한다.

XML 데이타 타입 인스턴스를 SQL에서 다루기 위하여 부가적으로 정의해야 하는 함수가 있다. SQL 질의에서 관계 데이타와 XML 데이타 간의 조인연산(join operation)이 발생할 때, 연산의 단위가 튜플(tuple)이므로, XML 데이타 타입 인스턴스에서 최상위 레벨(top-level) 엘리먼트가 하나의 튜플이 되도록 XML 데이타 타입 인스턴스를 분할하는 함수가 필요하다. 우리는 그 함수를 *XMLSEQUENCE()* 라 정의한다.

각 함수의 기능과 구문(syntax)를 정리하면 다음과 같다. (본 논문에 포함되지 않은 구문은 SQL:2003 표준을 준수한다.)

XMLELEMENT() : XML 데이타 타입 인스턴스와 XPath 표현식을 전달받아서 XML 단편이 포함된 XML

데이타 타입 인스턴스를 반환한다.

<XML extract> ::=
XMLEXTRACT <left paren> <xml value expression> <comma>
<path expression> <right paren>

XMLEXTRACTVALUE() : XML 데이타 타입 인스턴스와 XPath 표현식을 전달받아서 결과 노드의 스칼라 값을 반환한다.

<XML extract value> ::=
XMLEXTRACTVALUE <left paren> <xml value expression> <comma>
<path expression> <right paren>

XMLSEQUENCE() : XML 데이타 타입 인스턴스를 전달받아서 최상위-레벨 엘리먼트가 하나의 튜플을 이루는 XML 데이타 타입 튜플 집합을 반환한다.

<XML sequence> ::=
XMLSEQUENCE <left paren> <xml value expression> <comma>
<path expression> right paren>

SQL:2003을 지원하는 DBMS들은 위에서 정의한 사용자 정의 함수와 유사한 기능을 구현하고 있으므로, 구문은 다르더라도 유사한 기능을 수행하는 함수로 치환한다며, 본 논문에서 정의한 사용자 정의 함수를 포함한 SQL문은 DBMS 독립적으로 실행될 수 있다.

SQL 템플릿은 데이타 표현식을 사용하여 표현되며, 데이타 표현식에는 보조 구문의 데이타가 저장된다. 데이타 표현식의 구조를 자세히 살펴 보자.

데이타 표현식은 크게 다음과 같은 데이타 구조를 가진다.

_PathExpr : Path 표현식의 데이타 저장
_Step : Path 표현식의 각 Step의 데이타 저장
_Predicate : Path 표현식의 애트리뷰트 이름과 값 저장
_Attribute : 엘리먼트 생성시의 애트리뷰트 이름과 값 저장
_ComparisonExpr : 비교 표현식 저장

파싱된(parsing) Path 표현식의 데이타를 저장하기 위하여 *_PathExpr*를 정의한다. *_PathExpr*에는 변수 이름과 문자열 값, StepExpr의 리스트와 Predicate 값을 가진다. *_Step*은 Path 표현식의 각 Step를 저장하기 위한 데이타 구조이다. *_Step*에는 축(axis) 정보, 애트리뷰트 축 정보, 그리고, 적법한 이름(Qualified Name)이 저장된다. *_Predicate*에는 Path 표현식의 애트리뷰트 이름과 그 값이 저장된다. 엘리먼트를 생성할 때, 애트리뷰트 이름과 그 값을 전달하게 되는데, 이 때 *_Attribute*를 사용하게 된다. 비교 표현식을 저장하기 위하여 *_ComparisonExpr*를 정의한다. *_ComparisonExpr*에는 비교 연산자와 두 개의 Path 표현식이 저장된다.

_PathExpr
(VN: VarName, // Variable name
SL: StringLiteral, // Sstring literal
LS[]: List Of Step, // List of *_Step*
Pre: Predicate) // *_predicate*

_Step
(AX: Axis, // Axis ('/' |'//')
AT: At, // Attribute axis ([] | '@')
QN: QName) // Qualified name

_Predicate
(QN: QName, // Qualified predicate name
PE: PathExpr) // Path expression

_Attribute
(QN: QName, // Qualified predicate name
PE: PathExpr) // Path expression

_ComparisonExpr
(comp: Comparison Operator, // Comparison operator
p1: PathExpr, // Path expression
p2: PathExpr) // Path expression

Step 리스트를 담고 있는 LS[]에서 LS_0(숫자 "0"임. 즉, 첫 번째 리스트)는 표 2에서 유지하는 변수 정보를 이용하여 Path 표현식의 앞부분에 등장하는 변수에 대한 정보 즉, 엘리먼트 이름을 저장한다. Step 리스트 중 제일 마지막 리스트를 LS_k라고 지칭하기로 한다. 즉, 하나의 Path 표현식은 k개의 Step으로 구성되어 있다.

SQL 템플릿의 표현을 간략하게 하기 위하여 다음의 단축어를 사용한다.

CBN: Current Block Name
PBN: Previous Block Name
XT: XMLEXTRACT

XTV: XMLEXTRACTVALUE
XS: XMLSEQUENCE

XQuery 질의는 SQL의 WITH 절을 사용하여 연속적인 서브질의 블럭으로 변환되는데, *CBN*과 *PBN*은 질의 변환을 수행할 때 부여되는 서브질의 블럭 이름이다. *CBN*은 변환이 진행 중인 서브 질의 블럭 이름이고, *PBN*은 이전에 변환한 서브질의의 블럭 이름이다. 즉, *CBN*에서 *PBN*을 참조하며, *CBN*과 *PBN*은 자동적으로 생성되고 유지된다. *XT*, *XTV* 그리고 *XS*는 3.3절에서 정의한 함수를 지칭하기 위한 단축어이다.

본 논문에서 사용하게 될 XML 문서와 XQuery 질의, 그리고 질의 수행 결과는 그림 2와 같다. XQuery 및 XML 문서는 XMark[25]에서 가져온 것이다. 그림 2-b의 XQuery 질의는 경매가 종료된 경매 정보에서 사람마다 몇 개의 품목을 샀는가를 묻는 질의이다.

핵심 구문에 대응하는 SQL 템플릿의 리스트는 다음과 같다. (SQL 템플릿의 이름은 대응하는 XQuery 구문의 이름을 이탤릭체로 표현한다.)

FunctionCall
ForClause
LetClause
WhereClause
OrderByClause
ReturnClause
DirElemConstructor

3.3.1 *FunctionCall*

*FunctionCall*은 XQuery에 등장하는 다양한 함수들에 대한 SQL 템플릿을 제공한다. XQuery에서 정의하는 함수는 매우 다양하므로, 우리는 본 논문에서 모든 함수에 대한 SQL 템플릿을 정의하지 않는다. 함수를 SQL 템플릿으로 변환할 수 있음을 보여주기 위해 본 논문에

```
<?xml version="1.0" ?>
<site>
<regions>
<africa>
<item id="item0">
<location>United States</location>
<quantity>1</quantity>
<name>duteous nine eighteen </name>
<payment>Creditcard</payment>
<description>
//snip
```

a. XML 문서

```
1:  for $p in doc("auction.xml")/site/people/person
2:  let $a := for $t in doc("auction.xml")/site/closed_auctions/closed_auction
3:  where $t/buyer/@person = $p/@id
4:  return $t
5:  return <item person="{$p/name}"> {count ($a)} </item>
```

b. XQuery 질의

```
<item person="Masanao Marsiglia">1</item>
<item person="Kishor Monkewich">2</item>
<item person="Saul Schaap">2</item>
<item person="Laurian Grass">3</item>
<item person="Aloys Singleton">1</item>
<item person="Bassem Manderick">1</item>
```

c. 질의 수행 결과

그림 2 XML 문서, XQuery 질의 그리고 결과

서는 doc(), count()에 대한 SQL 템플릿을 정의한다.

XQuery 질의가 Parser에 의해 파싱이 되면, XQuery의 핵심 구문은 추상구문트리를 구성하고, 보조 구문은 데이타 표현식이라는 데이타 구조를 구성한다. XQuery의 doc 함수는 그림 3과 같은 데이타 표현식을 가지게 된다.

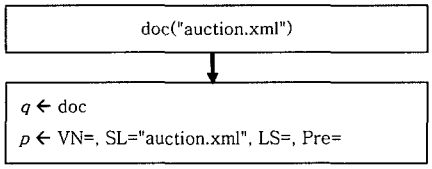

그림 3 XQuery의 doc 함수와 데이타 표현식

그림 3에서 q는 함수 이름에 해당하고, p는 path 표현식에 해당한다. 따라서, 이 데이타 표현식은 함수 이름이 "doc" 이고, SL(즉, String Literal)이 "auction.xml" 임을 나타낸다. q와 p는 SQL 템플릿인 *FunctionCall*의 인자로 전달되어, doc 함수를 SQL로 변환하는데 사용된다.

그림 4는 FunctionCall 의 SQL 템플릿을 나타낸다. 전달받은 q(라인 2)가 "doc"(라인 4) 일때는, 라인 5~8에 있는 SQL 템플릿이 사용되고, "count"(라인 9) 일때는 라인 10~14까지의 SQL 템플릿이 사용된다. q가 "doc" 일때의 SQL 템플릿을 살펴보자. 라인 8의 FROM 절에는 p.SL이 들어간다. 왜냐하면, doc 함수의 SL에는 XML 파일 이름이 입력되기 때문이다. 본 논문에서는 XML 파일 이름에 해당하는 XML 타입 뷰를 미리 생성해 두므로, 파일 이름을 바로 FROM 절에서 사용할 수 있다. 라인 5는 변환되는 SQL의 별명(alias)을 기술한다. 이는 XQuery의 각 핵심 구문이 WITH 절의 서브 블록으로 표현되기 때문에 모든 SQL 템플릿은 별명을 가지게 되고, *CBN*은 앞에서 기술한대로 자동적으로 부여된다. 라인 7의 SELECT 절에서 *XT* 함수를 사용해 XML 인스턴스를 추출한다. *XT* 함수의 전달인자로 '/'가 전달되는데, 이유는 XML 뷰에서 전체 XML 인스턴스를 추출하기 때문이다. 그리고, SELECT 절에서의 XML 인스턴스에 대한 칼럼명을 xvalue로 통일하기 위해 "as xvalue"라고 기술해 줌으로써, XQuery의 doc 함수를 SQL로 변환한다.

q가 *"count"*일 때 사용하는 SQL 템플릿은 라인 10~14에 나타나 있다. 라인 13의 FROM 절에는 p.VN이 사용된다. p.VN에는 변수가 저장되어 있고, 이 변수와 표 2를 이용하여 해당하는 서브 질의 블록 이름으로 대체하게 된다. 표 2에 나타난 정보의 생성 및 유지 방법은 제 3.3.2 절에서 자세히 설명한다. 라인 13~14를 살펴보면 *XT*를 사용하여 해당 path 표현식의 데이타들을 추출하고 이것을 최상위-레벨 엘리먼트들의 집합으로 만든 다음, 이를 table 함수를 사용하여 임시 테이블(v)을 만들고, v에 대해 count 함수를 적용하여 XQuery의 count함수를 SQL로 변환한다.

```
1:  FunctionCall:
2:  ( q: QName,              // Function name
3:    p: PathExpr )          // Path expression
4:  ⇒ if q equals "doc"
5:  CBN
6:  as
7:  SELECT XT(v0.xvalue, '/') as xvalue
8:  FROM p.SL v0
9:  ⇒ if q equals "count"
10: CBN
11: as
12: SELECT count(XT(value(v), 'p.LSk.QN')) as xvalue
13: FROM p.VN v0, table(XS(XT(v0.xvalue,
14: 'p.LS0.QN + ... + p.LSk.AX + p.LSk.AT + p.LSk.QN'))) v
```

그림 4 FunctionCall의 SQL 템플릿

그림 3의 XQuery는 그림 5와 같이 SQL 서브 블럭으로 변환된다. 그림 5의 FROM 절에 보이는 "auction"은 미리 정의된 XML 데이타 타입 뷰이다. XML 데이타 타입 뷰는 SQL/XML 출판 함수를 이용하여 쉽게 생성할 수 있다[4].

```
bnl as
 ( SELECT XMLEXTRACT(v0.xvalue, '/') as xvalue
   FROM auction v0
)
```

그림 5 doc 함수의 SQL 변환 결과

3.3.2 *ForClause*

XQuery의 for 절은 XML 데이타 타입 뷰로부터 Path 표현식에 해당하는 XML 부분을 반환하는데 사용된다. 전달인자로는 for 변수이름, Path 표현식이 전달된다.

```
1:  ForClause:
2:  ( var: VarName,          // Variable name
3:    p: PathExpr )          // Path expression
4:  ⇒
5:  CBN
6:  as
7:  SELECT XT(value(v), 'p.LSk.QN') as xvalue
8:  FROM p.VN v0, table(XS(XT(v0.xvalue,
9:  'p.LS0.QN + ... + p.LSk.AX + p.LSk.AT + p.LSk.QN'))) v
```

그림 6 ForClause의 SQL 템플릿

XQuery에서는 변수를 선언하여, 질의를 구성하는 다른 구문 내에서 이 변수를 사용할 수 있다. XQuery 구문 중에서 for 절과 let 절에서 새로운 변수를 선언할 수 있다. 그림 2의 예제 XQuery에서는 바깥쪽 FLWOR 표현식의 for 절(라인 1)에서 선언한 변수(p)가 안쪽 FLWOR 표현식의 where 절(라인 3)에서 사용되고 있다. 따라서, XQuery 질의를 SQL 질의로 변환하기 위해서는 이런 변수 정보를 유지할 필요가 있다. 우리는 표 2와 같이 정보를 유지한다. 유지하는 정보는 변수 이름(VarName), 변수에 해당하는 최상위 엘리먼트 이름(Element Name), 그리고, 그 변수에 대해 가장 나중에 변환된 서브 질의 블록 이름(CBN)이다. 가장 나중에 정의된 서브 질의 블록 이름을 유지하는 이유는 다음과 같다. 그림 2의 XQuery 질의에서 라인 2~4에 걸쳐 변수 t 가 사용되고 있다. 라인 2의 for 절에서 변수가 선언되면, 변수 t의 초기값은 라인 2의 for 절을 변환한 서브질의 블록이 된다. 그러나, 라인 3의 where 절에 의해 변수 t 가 가리키는 XML 인스턴스가 등호 조건에 의해 필터링 되므로, 이제 t의 값은 where 절에 의해 필터링 된 XML 인스턴스이고, 이는 where 절을 SQL로 변환한 서브 질의 블록이 된다. 따라서, 라인 4의 return 절에서 사용하는 변수 t는 라인 2의 for 절에 변환된 서브 질의 블록을 사용하는 것이 아니라, where 절에 의해 필터링 되고, 최신의 서브 질의 블록을 사용하게 되고 이렇게 하기 위해 같은 변수에 대해 가장 나중에 정의된 서브 질의 블록 이름을 유지하는 것이다.

표 2 질의 변수 정보

VarName	Element Name	CBN
a	paper	bn1
t	author	bn3
...	...	...

3.3.3 *LetClause*

XQuery 구문의 let 절은 for 절과 유사하나, 그 결과는 전혀 다르다. XQuery[8]의 Working Draft에 포함된 예제는 for 절과 let 절의 차이를 잘 설명해 준다. 그림 7의 라인 1~6은 for 절이 포함된 XQuery 질의이며, 라인 7~10은 let 절이 포함된 XQuery 질의이다. for 절에서 선언된 변수는 반복적으로 그 결과를 적용하며, let 절에서 선언된 변수에는 전체 결과가 바인딩 된다.

따라서, let 절을 SQL로 변환하기 위해서는 SQL 변환된 결과에 집합(aggregation) 함수를 적용시켜야 한다. 이를 위해 XQuery의 let 절에 대한 SQL 템플릿인 그림 8의 SELECT 절(라인 7)에 XMLAGG 함수를 적용시킨다.

```
1:  for $s in (<one/>, <two/>, <three/>)
2:  return <out>{$s}</out>
3:  ⇒
4:  <out><one/></out>
5:  <out><two/></out>
6:  <out><three/></out>
7:  let $s := (<one/>, <two/>, <three/>)
8:  return <out>{$s}</out>
9:  ⇒
10: <out><one/><two/><three/></out>
```

그림 7 각각 let 절과 for 절을 포함한 XQuery 및 결과

```
1:  LetClause:
2:  ( var: VarName,        // Variable name
3:  p: PathExpr)           // Path expression
4:  ⇒
5:  CBN
6:  as
7:  SELECT XMLAGG(XT(value(v), 'p.LSk.QN')) as xvalue
8:  FROM p.VN v0, table(XS(XT(v0.xvalue,
9:  'p.LS0.QN + ... + p.LSk.AX + p.LSk.AT + p.LSk.QN'))) v
```

그림 8 LetClause의 SQL 템플릿

3.3.4 *WhereClause*

WhereClause는 for 절이나 let 절에 의한 변수 바인딩이나 반복에 대해 조건에 따라 필터링하는 역할을 한다. *WhereClause* 템플릿은 SQL의 where 구문을 사용하여 템플릿을 생성한다. WhereClause를 위한 SQL 템플릿은 전달인자인 ComparisonExpr의 특성에 따라 여섯 가지로 나뉜다. ComparisonExpr의 첫 번째 Path 표현식은 마지막 Step이 엘리먼트 이름이나 애트리뷰트 이름을 가진 Path 표현식이 될 수 있고, 두 번째 Path 표현식은 마지막 Step이 엘리먼트 이름이나 애트리뷰트 이름을 가진 Path 표현식이거나, 문자열이 될 수 있다. 이를 표로 나타내면 표 3과 같다.

표 3 ComparisonExpr에서 PathExpr의 조합

		PathExpr 2		
		path/elem	path/att	string
PathExpr 1	path/elem	1)	2)	3)
	path/att	4)	5)	6)

표 3 에 나타나 있는 1) 부터 6) 까지의 번호는 각각의 조합의 경우를 나타낸다. 각각의 경우에 대한 SQL 템플릿은 다음과 같이 정의할 수 있다. 4), 5), 6)의 경우는 1), 2), 3)의 SQL 템플릿을 통해 쉽게 정의할 수

있으므로, 본 논문에서는 1), 2), 3)의 경우만 설명한다.

1) ComparisonExpr의 두 인자 모두 마지막 step 이 엘리먼트인 Path 표현식인 경우

```
1:  WhereClause:
2:  ( c: ComparisonExpr)  // Comparison expression
3:  ⇒
4:  CBN
5:  as
6:  SELECT XT(v0.xvalue, 'c.p1.LS0.QN') as xvalue
7:  FROM c.p1.VN v0, t able(XS(XT(v0.xvalue,
8:           'c.p1.LS0.QN +
9:           ... + c.p1.LSk.AX + c.p1.LSk.AT + c.p1.LSk.QN'))) v,
10: c.p2.VN v1, table(XS(XT(v1.xvalue,
11:          'c.p2.LS0.QN +
12:          ... + c.p2.LSk.AX + c.p2.LSk.AT + c.p2.LSk.QN'))) vv
13: WHERE XTV(value(v), 'c.p1.LSk.QN') c.comp
14: XTV(value(vv), 'c.p2.LSk.QN')
```

2) ComparisonExpr의 한쪽 인자는 마지막 step 이 엘리먼트인 Path 표현식이고, 다른 한쪽 인자는 마지막 step이 애트리뷰트인 Path 표현식인 경우

```
1:  WhereClause:
2:  ( c: ComparisonExpr)  // Comparison expression
3:  ⇒
4:  CBN
5:  as
6:  SELECT XT(v0.xvalue, 'c.p1.LS0.QN') as xvalue
7:  FROM c.p1.VN v0, table(XS(XT(v0.xvalue,
8:           'c.p1.LS0.QN +
9:           ... + c.p1.LSk.AX + c.p1.LSk.AT + c.p1.LSk.QN'))) v,
10: c.p2.VN v1, table(XS(XT(v1.xvalue,
11:          'c.p2.LS0.QN +
12:          ... + c.p2.LSk-1.AX + c.p2.LSk-1.AT + c.p2.LSk-1.QN'))) vv
13: WHERE XTV(value(v), 'c.p1.LSk.QN') c.comp
14: XTV(value(vv), 'c.p2.LSk.QN')
```

3) ComparisonExpr의 한쪽 인자는 마지막 step 이 엘리먼트인 Path 표현식이고, 다른 한쪽 인자는 String Literal인 경우

```
1:  WhereClause:
2:  ( c: ComparisonExpr)  // Comparison expression
3:  ⇒
4:  CBN
5:  as
6:  SELECT XT(v0.xvalue, 'c.p1.LS0.QN') as xvalue
7:  FROM c.p1.VN v0, table(XS(XT(v0.xvalue,
8:           'c.p1.LS0.QN +
9:           ... + c.p1.LSk.AX + c.p1.LSk.AT + c.p1.LSk.QN'))) v
10: WHERE XTV(value(v), 'c.p1.LSk.QN') c.comp'c.p2.SL'
```

3.3.5 *OrderByClause*

OrderByClause는 for 절이나 let 절에 의한 변수 바인딩이나 반복에 대해 옵션으로 where 절에 의해 필터링 된 튜플들에 대한 순서를 정한다. *OrderByClause* 템플릿은 SQL의 order by 구문을 사용하여 템플릿을 정의한다. OrderByClause는 전달인자인 PathExpr의 종류에 따라 두 가지의 SQL 템플릿을 가진다. 왜냐하면 PathExpr의 마지막 Step의 종류에 따라 XMLSEQUENCE 함수로 전달되는 전달인자가 달라지기 때문이다.

1) Path 표현식의 마지막 step이 엘리먼트인 경우

```
1:  OrderByClause:
2:  ( p: PathExpr )   // Path expression
3:  ⇒
4:  CBN
5:  as
6:  SELECT XT(v0.xvalue, 'p.LS0.QN') as xvalue
7:  FROM p.VN v0, table(XS(XT(v0.xvalue,
8:           'p.LS0.QN + ... + p.LSk.AX + p.LSk.AT + p.LSk.QN'))) v
9:  ORDER BY XTV(value(v), 'p.LSk.QN')
```

2) Path 표현식의 마지막 step 이 애트리뷰트인 경우

```
1:  OrderByClause:
2:  ( p: PathExpr )   // Path expression
3:  ⇒
4:  CBN
5:  as
6:  SELECT XT(v0.xvalue, 'p.LS0.QN') as xvalue
7:  FROM p.VN v0, table(XS(XT(v0.xvalue,
8:           'p.LS0.QN + ... + p.LSk-1.AX + p.LSk-1.AT + p.LSk-1.QN'))) v
9:  ORDER BY XTV(value(v), 'p.LSk.QN')
```

3.3.6 *ReturnClause*

*ReturnClause*는 where 절이나 order by 절을 통과한 튜플을 리턴한다.

```
1:  ReturnClause:
2:  ( p: PathExpr ) // Path expression
3:  ⇒
4:  CBN
5:  as
6:  SELECT XT(v0.xvalue,
7:  'p.LS0.QN + ... + p.LSk.AX + p.LSk.AT + p.LSk.QN') as xvalue
8:  FROM p.VN v0
```

3.3.7 *DirElemConstructor*

*DirElemConstructor*는 태그 이름과 애트리뷰트, 컨텐트를 전달 받아서 XML 데이타 타입 인스턴스를 생성한다.

```
1:  DirElemConstructor:
2:  ( t: TagName,       // Tag name
3:  att: Attribute      // Attribute
4:  p: PathExpr)        // Content
5:  ⇒
6:  CBN
7:  as
8:  SELECT XMLELEMENT(name "t",
9:              XMLATTRIBUTES(XTT(v0.xvalue, 'att.PE.LS0.QN +
10:                 ... + att.PE.LSk.AX + att.PE.LSk.AT + att.PE.LSk.QN')
11:                 as att.QN ),
12: XMLAGG(XTT(v1.xvalue,
13: 'p.LS0.QN + ...+ p.LSk.AX + p.LSk.AT + p.LSk.QN'))) as xvalue
14: FROM att.PE.VN v0, p.VN v1
15: WHERE v0.id = v1.id
```

3.4 XQuery to SQL 변환

본 절에서는 XQuery 질의의 SQL 변환을 예제를 통해 설명한다. XQuery 질의 및 XML 문서는 그림 2에 나타나 있다.

XQuery를 변환하는 방법으로, 우선 XQuery 표현식을 간략화 하기 위하여 XQuery 정규화가 사용된다. 예를 들면, for 절에 두 개 이상의 변수 반복이 있을 때, 정규화를 거쳐 중첩된 for 절로 표현하면서 하나의 for 절에는 하나의 변수만 존재하게 하는 것이다. XQuery의 정규화에 관한 연구는 [23], [14] 등에서 연구되었다. XQuery 질의가 질의 번역기에 들어 오기 전에 정규화를 선행하도록 한다.

정규화된 XQuery 질의를 분석하여 추상구문트리(Abstract Syntax Tree)를 생성한다. 추상구문트리는 JavaCC라는 도구와 XQuery BNF를 이용하여 만든 Parser에 생성되며, 구체적인 구현 방법은 제 4절에서 설명한다.

추상구문트리는 Transformer에 의해 핵심 구문을 중심으로 재구성된다. Transformer는 추상구문트리를 전위 순회방식으로 순회하면서, 보조구문을 가진 노드를 데이타 표현식으로 재구성하고, 이를 핵심구문을 가진 부모나 조상 노드의 속성에 포함시켜서 추상구문트리의 노드가 핵심구문만으로 이루어 지도록 재구성한다. 즉, 그림 9에서 보는 것처럼, 핵심구문1,2와 보조구문1,2,3을 가진 추상구문트리를 핵심구문1과 핵심구문2가 트리를 구성하고, 보조구문1,2,3은 각각 부모 노드의 속성으로 변환한다.

추상구문트리를 재구성하는 알고리즘이 그림 10에 나타나 있다. transformAST 함수는 재구성하고자 하는 추상구문트리(a_1)와 재구성된 추상구문트리가 저장될 데이타구조(t_1)를 전달받아서 알고리즘을 수행한다. 이 알고리즘은 a_1이 자식 노드를 가질 때(라인 5) 수행된다. a_1이 자식 노드를 가지면, 각각의 자식 노드에 대해 그 노드가 핵심구문인지를 검사하고(라인 7) 핵심구문이면 라인 8~10를 수행하고, 재귀적으로 변환 함수를 호출한다(라인 11). 이때, 전달인자는 새로운 재구성된 추상구문트리(t_2)와 추상구문트리 a_1의 서브트리인 a_2이다. 그리고, a_2가 핵심구문이 아닐 때는 a_2의 자식 노드 a_3에 대해 핵심구문을 만날 때 까지 데이타 표현식 d를 생성한다(라인 15). 여기서 생성되는 데이타 표현식은 제 3.3 절에 정의되어 있다.

그림 2의 XQuery 질의를 핵심 구문을 중심으로 재구성하면, 그림 11과 같다. (데이타 표현식은 지면 제약상

```
1:  procedure transformAST(transformed AST t1, AST a1) {
2:    // t1,t2 : node of transformed AST
3:    // a1,a2,a3 : node of AST
4:    // d : specific data structure
5:    if (a1 has child node) {
6:      for each a1's child node a2 {
7:        if(a2 is Core Syntax) {
8:          Generate transformed AST t2
9:          Add t2 to attribute of d
10:         Add t2 to child node of t1
11:         transformAST(t2, a2)
12:       } else {
13:       for each a2's child node a3 {
14:         while (a3 is not Core Syntax) {
15:           Generate d
16:         }
17:         transformAST(t1, a3)
18:       } } } } }
```

그림 10 추상구문트리를 재구성하는 알고리즘

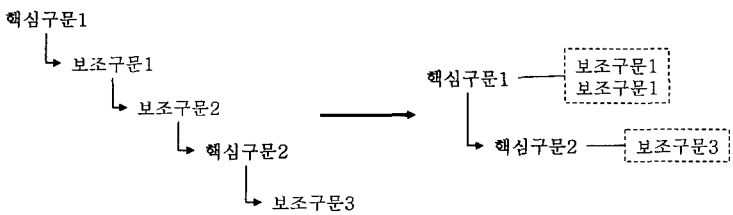

그림 9 추상구문트리(왼쪽)와 재구성된 추상구문트리(오른쪽)

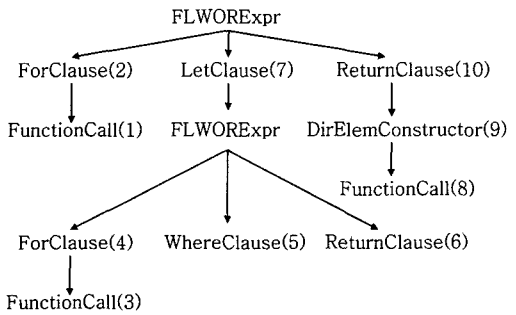

그림 11 재구성된 추상구문트리

표현하지 않았음.)

재구성된 추상구문트리는 전위(preorder) 순회 방식으로 각각의 노드를 탐색을 하면서 각각의 구문을 대응하는 SQL 템플릿으로 치환된다. 즉, 그림 11의 추상구문트리는 각 노드의 괄호 안에 표시된 번호 순서대로 SQL 템플릿으로 치환되어 최종적으로 단일 SQL 질의가 생성된다. SQL로 변환하는 알고리즘은 그림 12에 나타나 있다. generateSQL 함수는 재구성된 추상구문트리 t_1을 전달 받아 SQL을 생성한다. t_1이 자식을 가지고 있을 때, 그 노드들을 전위 순회 방식으로 각 노드를 방문 하면서, 노드 이름에 따라 해당하는 SQL 템플릿을 적용한다(라인 7~11).

```
1:  procedure generateSQL(transformed AST t1) {
2:    if(t1 has child node) {
3:      for each t1's child node t2 {
4:        generateSQL(t2)
5:      }
6:      s ← node name of t2
7:      if(s equals 'FunctionCall')
8:        Use SQL template of Fig. 4
9:      else if(s equals 'ForClause')
10:       Use SQL template of Fig. 6
11:     ....
12:   }   }
```

그림 12 SQL 생성 알고리즘

그림 12의 알고리즘을 적용하여 그림 2의 XQuery 질의를 SQL로 변환하면, 그 결과는 그림 13과 같다.

4. 질의 번역기 설계 및 구현

본 장에서는 XQuery 질의를 SQL 질의로 변환하는 질의 번역기에 대해 설명한다. 제 4.1절에서는 질의 번역기의 전체적인 구조를 설명하고, 제 4.2절에서는 프로

```
1:  with
2:  bn1 as          // FunctionCall[doc]
3:  ( select EXTRACT(v0.xvalue, '/') as xvalue
4:    from auction v0
5:  ) , bn2 as          // ForClause
6:  ( select EXTRACT(value(v), 'person') as xvalue
7:    from bn1 v0, table(xmlsequence(EXTRACT(v0.xvalue,
8:      '/site/people/person'))) v
9:  ), bn3 as          // FunctionCall[doc]
10: ( select EXTRACT(v0.xvalue, '/site') as xvalue
11: from auction v0
12: ), bn4 as          // ForClause
13: ( select EXTRACT(value(v), 'closed_auction') as xvalue
14: from bn3 v0, table(xmlsequence(EXTRACT(v0.xvalue,
15: 'site/closed_auctions/closed_auction'))) v
16: ), bn5 as          // WhereClause
17: ( select EXTRACTVALUE(value(vv), 'person/@id') as aggkey,
18: EXTRACT(v0.xvalue, 'closed_auction') as xvalue
19: from bn4 v0, table(xmlsequence(EXTRACT(v0.xvalue,
20: 'closed_auction'))) v,
21: bn2 v1, table(xmlsequence(EXTRACT(v1.xvalue, 'person'))) vv
22: where EXTRACTVALUE(value(v), 'closed_auction/buyer/@person')
23: = EXTRACTVALUE(value(vv), 'person/@id')
24: ), bn6 as          // ReturnClause
25: ( select aggkey, EXTRACT(v0.xvalue, 'closed_auction') as xvalue
26: from bn5 v0
27: ), bn7 as          // LetClause
28: ( select aggkey, XMLAGG(EXTRACT(value(v), 'closed_auction'))
29: as xvalue
30: from bn6 v0, table(xmlsequence(EXTRACT(v0.xvalue,
31: 'closed_auction'))) v
32: group by aggkey
33: )// snip
34: select xvalue from bn10 v0;
```

그림 13 SQL 변환 결과 발췌 화면

토타입의 구현에 대해 설명한다.

4.1 전체 구조

질의 번역기의 전체적인 구조는 그림 14와 같다.

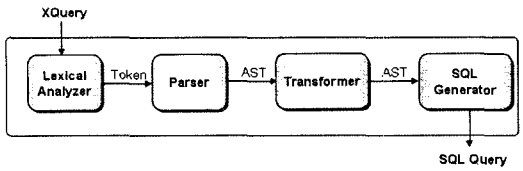

그림 14 XQuery to SQL 질의 번역기 구조

질의 번역기는 XQuery 질의를 입력 받아 SQL질의를 리턴한다. 질의 번역기는 크게 4개의 컴포넌트로 이루어져 있다. 각 컴포넌트의 기능은 다음과 같다.

- Lexical Analyzer - 입력 받은 XQuery 질의를 분석하여 XQuery 언어의 구문 단위로 토큰(token)을 생성한다.
- Parser - XQuery 언어의 구문을 분석하여 추상구문트리(Abstract Syntax Tree)를 생성한다.

- Transformer - 추상구문트리를 핵심 구문을 중심으로 재구성한다.
- SQL Generation - 추상구문트리를 단일 SQL로 변환한다.

4.2 프로토타입 구현

질의 번역기를 구현하기 위해서 우선 XQuery 질의에 대한 구문 분석을 하고, 파싱하는 파서가 필요하다. 본 논문에서는 XQuery BNF를 JavaCC라는 도구를 이용하여 Lexical Analyzer와 Parser의 틀을 구성하였다.

XQuery 질의를 입력받으면 Lexical Analyzer가 질의를 파싱하여 토큰과 요소들을 구분하고, Parser가 파싱된 XQuery 구문으로 추상구문트리를 구성한다. 그림 15은 Parser의 결과로 생성된 추상구문트리이다.

Transformer는 Parser에 의해 생성된 추상구문트리를 변경하여, XQuery의 핵심 구문만 노드가 되도록 재구성한다. 그림 16은 Transformer에 의해 재구성된 추

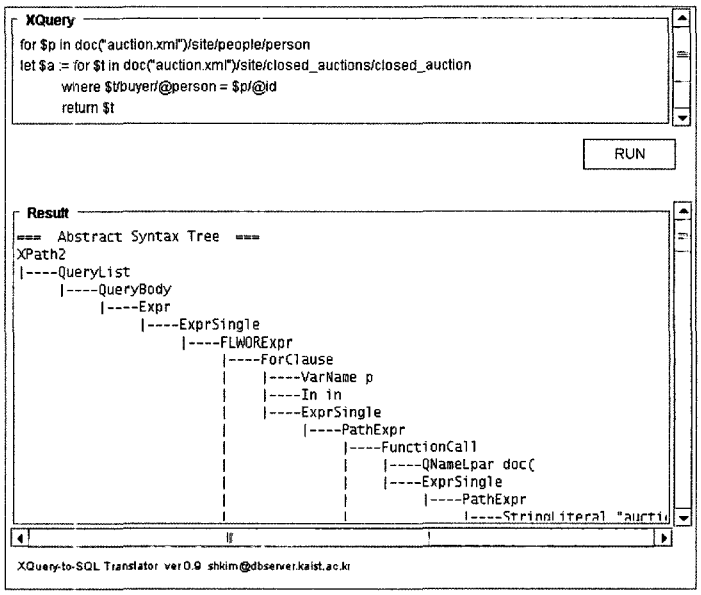

그림 15 Parser에 의해 생성된 추상구문트리 발췌화면

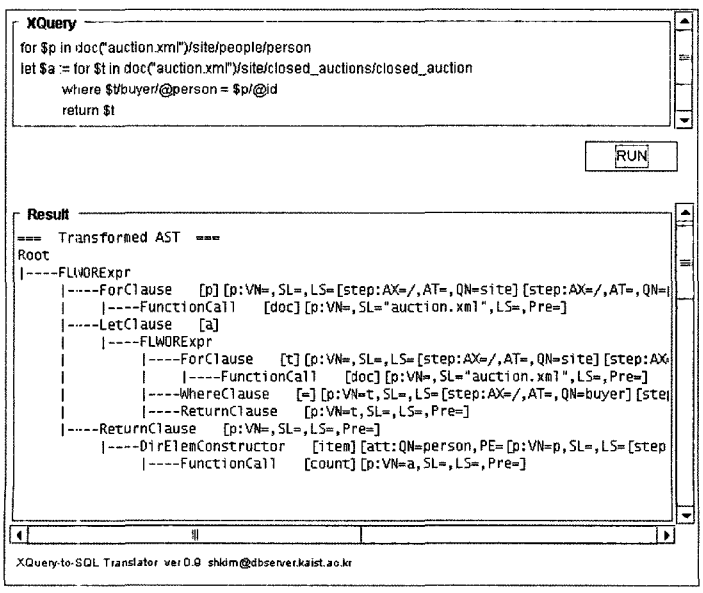

그림 16 Transformer에 의해 재구성된 추상구문트리 발췌화면

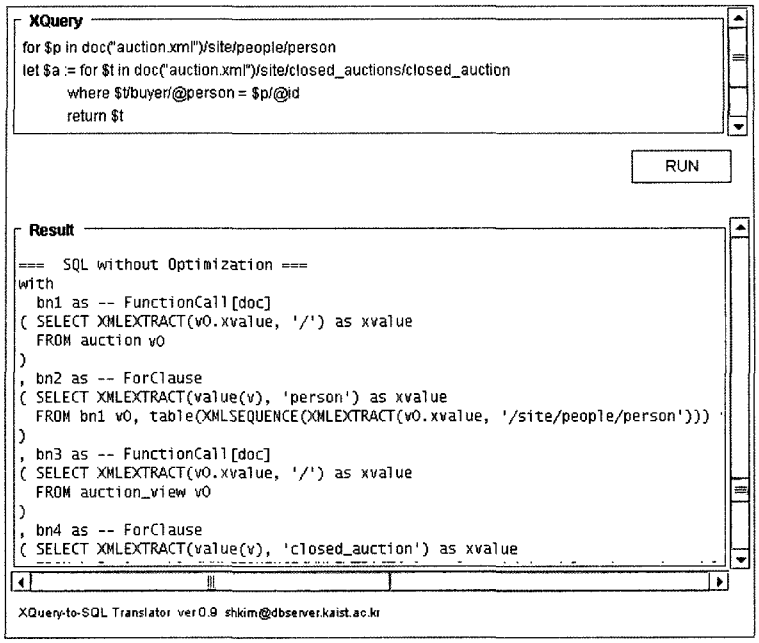

그림 17 XQuery to SQL 질의 번역기 수행 모습

상구문트리이다.

그리고, SQL Generator가 재구성된 추상구문트리를 전위 순회 탐색을 하면서 단일 SQL 질의를 생성한다.

그림 17은 XQuery 예제 1을 XQuery to SQL 질의 번역기를 사용하여 SQL로 변환한 모습이다.

5. 결 론

XML은 플랫폼 독립적이고, 반구조적인 특징으로 인해 여러 분야에서 활용되어 XML 데이타의 양이 많아졌다. 이에 따라, 대량의 XML 데이타를 저장하고 질의하는 방법에 대한 연구가 활발히 이루어졌다. XML 데이타를 저장하는 방법으로 관계형 데이타 베이스 시스템을 이용하는 방법이 널리 연구되고 있고, 관계형 데이타베이스에 대한 표준 질의 언어인 SQL도 XML을 지원하기 위하여 SQL:2003 표준에서는 SQL/XML 이라 불리는 XML을 지원 특징을 표준으로 포함하고 있다.

본 논문에서는 XML 데이타가 SQL:2003 표준을 지원하는 관계형 데이타베이스 시스템에 저장되어 있을 때, 이에 대한 XQuery 질의를 처리하기 위하여 SQL:2003 표준에서 정의하고 있는 SQL/XML과 사용자 정의 함수를 사용하여 XQuery 질의를 SQL 질의로 변환하는 번역기를 설계 및 구현하였다.

본 논문의 주요 공헌은 다음과 같다. 첫째, SQL:2003 표준에서 정의하고 있는 XML 지원 특징을 살펴보고, 표준에서 빠져 있는 XML 데이타 타입에서 데이타를 추출하는 함수를 제안하였다. 둘째, SQL:2003 표준과 사용자 정의 함수를 사용하여 SQL 템플릿을 정의하고, 정의된 템플릿을 기반으로 XQuery 질의를 SQL 질의로 변환하는 방법을 제안하였다. 셋째, 질의 번역기에 대한 설계와 구현을 자세히 기술하여 번역기의 가능성을 보였다.

참 고 문 헌

[1] T. Bray, et al., "Extensible Markup Language (XML) 1.1 W3C Recommendation," http://www.w3.org/TR/xml11, 2004.

[2] ISO/IEC 9075:1999 Information technology- Database languages - SQL.

[3] ISO/IEC 9075:2003 Information technology - Database languages - SQL (15 December 2003).

[4] D. Chamberlin, J. Robie, and D. Florescu, "Quilt: An XML Query Language for Heterogeneous Data Sources," Proceedings of WebDB, pp. 53-62, 2000.

[5] J. Robie, J. Lapp, and D. Schach., "XML Query Language (XQL)," Proceedings of QL, 1998.

[6] A. Deutsch, et al., "XML-QL: A Query Language for XML," Proceedings of WWW, 1999.

[7] J. Clark, et al., "XML Path Language (XPath) Version 1.0 W3C Recommendation," http://www.w3.org/TR/xpath, 1999.

[8] S. Boag, et al., "XQuery 1.0: An XML Query Language W3C Candidate Recommendation," http://www.w3.org/TR/xquery, 2005.

[9] J. Clark., "XSL Transformations (XSLT) 1.0 W3C Recommendation," http://www.w3.org/TR/xslt, 1999.
[10] S. DeRose, et al., "XML Pointer Lanuges (XPointer) W3C Working Draft," http://www.w3.org/TR/xptr, 2002.
[11] J. Shanmugasundaram, et al., "Querying XML Views of Relational Data," Proceedings of VLDB, pp. 261-270, 2001.
[12 I. Manolescu, D. Florescu, and D. Kossman., "Answering XML queries over heterogeneous data sources," Proceedings of VLDB, 2001.
[13] A. Deutsch and V. Tannen., "MARS: A System for Publishing XML from Mixed and Redundant Storage," Proceedings of VLDB, 2003.
[14] M. Fernandez, A. Morishima, and D. Suciu., "Efficient Evaluation of XML Middle-ware Queries," Proceedings of SIGMOD, pp. 103-114, 2002.
[15] X. Zhang, B. Pielech, and E. Rundesnteiner., "I Shrunk the XQuery! - An XML Algebra Optimization Approach," Proceedings of WIDM, pp. 15-22, 2002.
[16] D. DeHaan, D. Toman, M.P. Consens, and M.T. A Oszu., "A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding," Proceedings of SIGMOD, pp. 623-634, 2003.
[17] T. Grust, S. Sakr, J. Teubner., "XQuery on SQL Hosts," Proceedings of VLDB, pp. 252-263, 2004.
[18] D. Florescu, D. Kossman., "Storing and Querying XML Data using an RDMBS," IEEE Data Engineering Bulletin 22(3), pp. 27-34, 1999.
[19] F. Tian, et al., "The Design and Performance Evaluation of Alternative XML Storage Strategies," SIGMOD Record 31(1), pp. 5-10, 2002.
[20] ISO/IEC 9075-14:2003 Part 14: XML-Related Specifications (SQL/XML).
[21] G. Lee., "Mastering XML DB Queries in Oracle Database 10g Release 2 White Paper," http://www.oracle.com, 2003.
[22] M. Nicola and B. Linda., "Native XML Support in DB2 Universal Database," Proceedings of VLDB, pp. 1164-1174, 2005.
[23] D. Draper, et al., "XQuery 1.0 and XPath 2.0 Formal Semantics W3C Candidate Recommendations," http://www.w3.org/TR/xquery-semantics, 2005.
[24] R. Krishnamurthy, R. Kaushik, and J. Naughton., "XML-to-SQL Query Translation Literature: The State of the Art and Open Problems," Proceedings of XSym, pp. 1-18, 2003.
[25] R. Busse, et al., "XMark - An XML benchmark project," http://www.xml-benchmark.org

김 송 현

2000년 공군사관학교 전산학과 졸업 2005년 한국과학기술원 전산학과에서 석사학위 취득. 2005년~현재 공군사관학교 전산학과 교수. 관심분야는 XML Database, Database Systems 등

박 영 섭

2001년 한국과학기술원 전산학과 졸업 2003년 한국과학기술원 전산학과에서 석사학위 취득. 2003년~현재 한국과학기술원 전산학 박사과정. 관심분야는 XML 문서저장 및 질의처리 등

이 윤 준

1977년 서울대학교 계산통계학과 졸업 1979년 한국과학기술원 전산학과에서 석사학위 취득. 1983년 France, INPGEN-SIMAG에서 박사학위 취득. 1983년~1984년 France, IMAG 연구원. 1984년~현재 한국과학기술원 전산학과 교수 1989년 MCC(미) 초빙연구원. 1990년 CRIN(프) 객원교수 관심분야는 데이타베이스 시스템, 정보검색, 실시간 데이타베이스 등