XML 포함질의를 위한 확장형 인덱스

이 상 원†

요 약

XML 문서에 대한 포함질의는 XML의 핵심 질의 중의 하나이다. 따라서, XML 데이터를 지원하는 DBMS에서 이러 유형의 질의를 효과적으로 처리하는 것은 매우 중요한 문제이다. 최근 들어, 객체관계형 DBMS에 XML 데이터를 저장하려는 많은 노력들이 시도되고 있다. 본 논문에서는 객체관계형 DBMS에 BLOB 형태로 저장된 XML 데이터를 대상으로 포함질의를 효과적으로 처리하기 위한 확장형 인덱스를 제시한다. 즉, 객체관계형 DBMS의 확장성을 이용해서 포함질의 처리를 위한 효과적으로 처리하는 확장형 인덱스의 구현과 이 인덱스의 사용 방법을 기술한다.

An Extensible Index for XML Containment Queries

Sang-Won Lee†

ABSTRACT

Containment queries for XML documents is one of the most important query types, and thus the efficient support for this type of query is crucial for XML databases. Recently, object-relational database management system (ORDBMS) vendors try to store and retrieve XML data in their products. In this paper, we propose an extensible index to support containment queries over the XML data stored as BLOB type in ORDBMSs. That is, we describe how to implement the index using the extensibility feature of an ORDBMS, and describe its usage.

키워드 : XML 데이터(XML Data), 포함질의(Containment Queries), 객체관계형 DBMS(ORDBMS), 확장성(Extensibility), 확장형 인덱스(Extensible Index)

1. 서 론

XML[1]은 빠른 속도로 웹 상에서 데이터 표현(data representation)과 문서 교환(data exchange)의 표준으로 지정되고 있으며, 실제로 많은 회사나 단체들이 그들의 문서 포맷으로 XML을 채택하기 시작했다. XML은 앞으로 자동차 회사에서 각종 자동차에 대한 정보를 나타내기 위해 사용될 수 있고, 도서관에서 도서목록에 대한 정보를 나타내기 위해 사용될 수 있다. 또한 전자상거래 사이트에서 각종 상품에 대한 정보를 나타내기 위해 사용될 수 있다. 결국, 이러한 추세로 방대한 XML 문서가 생성되며, 이로 인해 XML 문서 데이터베이스로부터 정보를 효율적으로 추출하는 방안에 관한 연구가 필요하게 되었다.

이를 위해 W3C에서는 XML 문서들로부터 정보를 추출하기 위한 표준 질의어로 XQuery[2]를 채택했다. 그런데, 이 XQuery를 이용해서 XML 문서로부터 정보를 추출하는 경우, 성능 측면에서 XQuery의 핵심은 포함질의(containment query)의 처리 방법이다. 여기서 말하는 포함질의란 [3], XML 문서상에서 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들 간의 포함(containment) 관계에 기반을 둔 질의를 말한다. 이러한 포함질의가 XML 정보검색 시스템 상에서 XML 문서에 대한 질의의 핵심적인 부분이 되기 때문에 포함질의를 어떤 방법으로 지원하느냐 하는 것이 성능상의 중요한 문제가 된다.

본 논문에서는 객체관계형 DBMS에 저장된 XML 데이터에 대한 포함질의의 효과적인 처리를 위한 확장형 인덱스(extensible index)를 제안한다. 우선 XML 데이터의 저장 방안으로 XML 전용 데이터베이스(Native XML database)가 사용되기도 하지만, 지난 20년간 획기적으로 발달한 객체관계형 DBMS(Object-Relational DBMS, 이하 ORDBMS라 칭함)의 안전성, 질의 최적화, SQL 등의 장점으로 인해 앞으로는 ORDBMS가 XML 데이터 저장을 위한 주류적인 방법이 될 것이다[3, 4]. 따라서, ORDBMS상에서 포함질의의 효과적인 처리를 위해서는 XML 데이터 타입을 위한 새

※ 이 논문은 2002년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2002-003-D00447).
† 정 회 원 : 성균관대학교 정보통신공학부 교수
논문접수 : 2003년 10월 13일, 심사완료 : 2003년 12월 24일

로운 인덱스가 필요하다. 이를 위해 본 논문에서는 ORDBMS의 확장성(extensibility) 기능을 이용해서 XML 데이터에 대한 효과적인 포함질의 처리를 위한 확장형 인덱스(extensible index)를 제안한다.

논문의 기여사항 : 본 논문의 기여사항은 본 저자들이 참고문헌 [6]에서 제시한 "*관계형 테이블 기반의 역색인 인덱스 기법을 현재 상용 ORDBMS의 확장형 인덱스 기능을 활용해서 구현 가능하고 실용적으로 사용할 수 있음을 보이는 것*"이다. 관계형 테이블 기반의 확장형 인덱스 기법이 성능이 우수하나, 일반 사용자 입장에서 사용하려면 프로그램이나 도구의 도움을 받아서, 수동으로 인덱스용 테이블들을 생성하고, 이 테이블들의 구조를 파악해서 포함 질의를 복잡하게 작성해야만 하는 번거로움이 있다. 이를 해결하기 위한 방안으로 본 논문에서는 최근에 상용 ORDBMS에서 제공되기 시작한 확장형 인덱스 기능을 활용해서, XML 포함질의를 위한 확장형 인덱스를 고려하고 이를 구현하는 방법을 제시하게 되었다. 이렇게 구현된 XML 포함질의를 위한 확장형 인덱스를 사용하면, 사용자는 B+ 트리나 비트맵(bitmap) 인덱스와 똑 같은 방법으로, XML 포함질의 처리를 위한 인덱스를 손쉽게 생성할 수 있고, 사용자는 XPath 형태로 포함질의를 표현하면 확장형 인덱스에서 내부적으로 이를 인덱스용 테이블에 대한 질의로 자동변환해서 조건을 만족하는 문서들만 효율적으로 찾아준다.

본 논문의 구성은 다음과 같다. 먼저, 2장에서 예비 지식으로 XML의 포함질의, XML 포함질의를 위해 역색인 인덱스 기법에 기반한 포함질의 처리 방안, 그리고 ORDBMS의 확장성 기능을 소개한다. 3장에서는 대표적인 상용 ORDBMS의 확장성 기능을 이용해서 포함질의를 위한 확장형 인덱스의 구현 내용을 설명한다. 4장에서는 구현된 인덱스를 이용해서 XML 데이터에 대해 인덱스를 생성하고, 실제 SQL 질의가 수행되었을 때 이 인덱스를 이용해서 질의가 내부적으로 처리되는 과정을 보이겠다. 마지막으로 5장에서 결론과 향후 연구 과제를 기술하겠다.

한편, 기존 연구들, 특히 다양한 XML 인덱싱 기법들과의 차별화나 성능 평가는 본 논문의 핵심 내용이 아니다. 기존 연구와 우리가 제안한 인덱스 방법간의 차이점이나 성능 평가에 관한 자세한 내용은 참고 문헌[6]을 보기 바란다. 또한, 본 논문에는 관련 연구에 대한 절은 따로 두지를 않았는데, 그 이유는 우선 ORDBMS의 확장성을 이용한 포함질의를 위한 확장형 인덱스의 구현에 관한 연구가 거의 없고, 상용 ORDBMS들의 경우에도 자사의 포함질의 지원과 관련해서 구현 방안에 대해서는 공개된 자료가 없기 때문이다.

2. XML 포함질의를 위한 역색인 인덱스와 ORDBMS 확장성

2.1 XML 포함질의

포함질의는 XML 문서를 구성하고 있는 엘리먼트(elements), 애트리뷰트(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계에 기반을 둔 질의를 말한다. 이러한 포함질의는 XML 정보검색 시스템에 의해 처리되는 질의의 핵심이다. 본 논문에서는 포함질의를 나타내기 위해 XQuery[2]와 비슷한 경로 표현식을 사용한다.

본 논문에서는 기본 포함질의의 종류를 [4]에서와 같이 크게 다음 네 가지로 분류한다. (그림 1)의 XML문서에 대한 질의 예제로 살펴본다.

```
<Companies>
  <Company>
    <Symbol>AAPL</Symbol>
    <Name>Apple Computer, Inc.</Name>
    <Profile>
     <Address>
        <State>Texas</State>
        <City>Austin</City>
      </Address>
      <Description> Designs, develops, produces, markets and
       services
       microprocessor based personal computers, related
       software and peripheral
       products, including laser printers, scanners, compact
       disk read-only memory
       drives and other related products </Description>
    </Profile>
  </Company>
</Companies>
```

(그림 1) 예제 XML 문서

① **간접 포함질의** : 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계가 간접 포함관계(조상-자손 관계)로만 이루어진 질의. 예 : /companies//profile//'scanners'

맨 앞의 "/"는 companies가 문서의 루트 엘리먼트라는 것을 의미하고 "//"는 간접 포함관계(조상-자손 관계)를 의미하므로 "companies" 루트 엘리먼트가 자손 엘리먼트로 "profile"을 가지며 "profile" 엘리먼트의 자손 엘리먼트들의 내용(content) 중에 "scanners" 단어가 있는 XML 문서를 추출하라는 질의이다.

② **직접 포함질의** : 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계가 직접 포함관계(부모-자식 관계)로만 이루어진 질의. 예 : /companies/company/profile/description/'printers'

루트 엘리먼트 "companies"는 자식 엘리먼트로 "company"를 가지며 "company"는 자식 엘리먼트로 "profile"을 가진다. "profile"은 자식 엘리먼트로 "description"을 가지고 "description"은 그것의 내용(content)에 "printers"이라는 단어를 포함해야 한다. 결국, 위 조건들을 만족하는 XML 문서를 추출하라는 질의이다.

③ **완전 포함질의 :** 엘리먼트들(elements), 애트리뷰트들(attributes), 그리고 그것들의 내용을 이루는 텍스트 단어들간의 포함관계가 완전 포함관계로만 이루어진 질의. 예 : //symbol = 'AAPL'

"symbol" 엘리먼트가 "AAPL" 단어를 완전포함("symbol" 엘리먼트의 내용이 "AAPL" 단어만 포함하고 있는 경우)하고 있는 XML 문서를 검색하라는 질의이다.

④ **k-근접 포함질의 :** 텍스트 단어간의 근접도(proximity)에 기반을 둔 질의. 예 : Distance("printers", "scanners") ≤ 3 (K = 3)

"printers"와 "scanners"가 단위 거리로 3이내에 있는 XML 문서를 추출하라는 질의이다.

XML 정보검색 시스템 상에서의 질의의 대부분은 위의 네 가지 기본 포함질의들이 혼합된 형태가 핵심을 이룬다고 볼 수 있다. 예를 들면, 간접과 직접 포함질의가 혼합된 예로서 "/companies/company//description/"plastic'"을 둘 수 있고 간접, 직접, 그리고 완전 포함 질의가 혼합된 예로 "/companies/company//address/city='Austin'"을 들 수 있다.

2.2 포함질의 처리를 위한 관계형 테이블을 이용한 역색인의 확장 방안

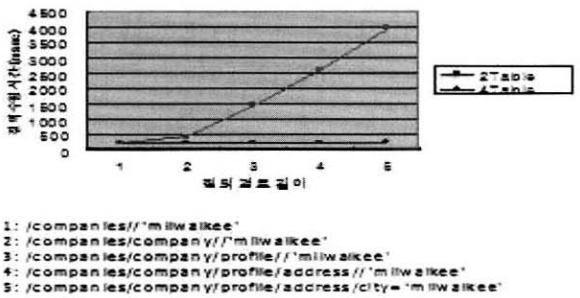
역색인은 정보검색 시스템에서 가장 널리 쓰이는 색인 방법으로 탐색 작업의 속도를 향상시키기 위해 텍스트에 대한 색인을 만들기 위한 단어기반 메커니즘이다[5]. 이 역색인 기법을 확장해서 XML 포함 질의를 위해 다음과 같은 두 개의 관계형 테이블로 모델링하는 방법을 [4]에서 최초로 제안했었다.

```
Elements (term, docno, begin, end, level)
Texts (term, docno, wordno, level)
```

그런데, 이 방법은 각각의 두 개의 엘리먼트간의 포함관계를 처리하기 위해 Elements 테이블 상에서 한 번의 셀프 조인(self-join)을 유발하고, 따라서 두 가지 큰 문제들을 야기한다[6]. 우선, 포함질의의 경로길이(path length)에 비례해서 조인 횟수가 증가한다. 두 번째 문제점은, 방대한 XML문서에 대해서 위와 같은 역색인 방식이 쓰이게 될 경우, Elements 테이블과 Texts 테이블은 크기가 매우 커지게 되므로 조인 연산 시, 항상 크기가 큰 테이블간의 조인이 발생한다는 점이다. 우리는 이 두 가지 문제점을 해결하기 위해, 역색인을 다음과 같은 4개의 테이블에 사상(mapping)시키는 방법을 사용했다.

```
Path (path, pathID)
PathIndex (pathID, docID, begin, end)
Term (term, termID)
TermIndex (termID, docID, pathID, position)
```

우리는 이 방법을 통해서 포함질의를 만족하는 XML 문서를 검색하는 성능을 획기적으로 향상할 수 있음을 보였다[6]. 우리가 제안한 방법과 기존 방법의 성능분석에 관한 자세한 내용은 [6]을 참고하기 바란다. (그림 2)는 [4]의 방법에 비해 우리가 제안한 방법이, "질의 경로의 길이"에 비례해서 획기적으로 우수함을 보이고 있다. 이와 유사한 연구 결과는 [7]에서도 확인할 수 있다.

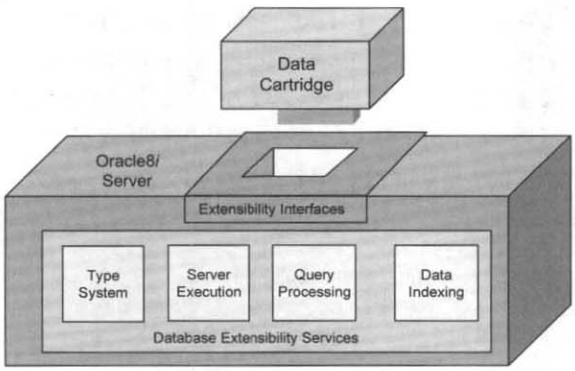


(그림 2) 질의 경로길이에 따른 질의 수행시간 변화

2.3 객체관계형 DBMS의 확장성

1990년대 초반에 관계형 DBMS 벤더들은 자사의 제품에 객체지향 개념을 지원하기 위한 많은 노력을 들였고, 현재 IBM DB2와 Oracle 9i 등을 포함한 주요한 관계형 DBMS들은 객체(object) 개념을 완전히 지원하고 있으며, 이들 DBMS를 일반적으로 ORDBMS라 부른다.

데이터베이스에서 객체 개념을 지원하는 주요 이유 중의 하나는 공간 데이터, 텍스트, 비디오, XML 등 새로운 멀티미디어 데이터 타입들을 필요로 하는 응용 분야를 효과적으로 지원하기 위해서이다. 이를 위해서 새로운 데이터 타입을 사용자가 DBMS에 등록할 수 있어야 하고, DBMS에서 이들 타입들에 필요한 연산들과 인덱싱 기능을 효과적으로 지원할 수 있어야 한다. 이처럼 ORDBMS에서 새로운 데이터 타입의 생성, 연산자 등록, 그리고 새로운 데이터 타입과 연산자를 위한 인덱스 기능을 추가할 수 있는 구현 인프라를 확장성(extensibility)이라 부른다[8]. 현재 주요한 ORDBMS 벤더들은 자사의 제품들에서 이 확장성 기능을 제공하고 있다[9-11].



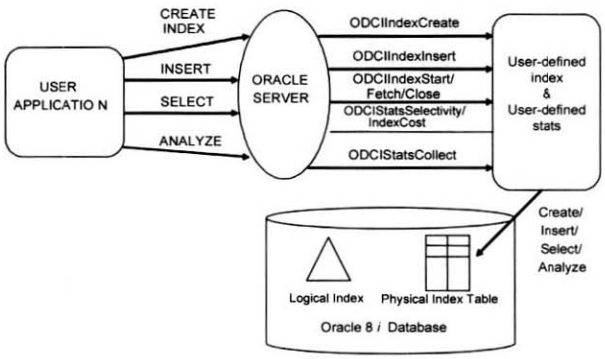
(그림 3) ORDBMS에서 확장성의 개념

(그림 3)에서처럼, 오라클사는 Data Catridge라는 이름으로 확장성 기능을 제공하고 있는데[9], 확장성 기능은 정수나 문자열과 같은 내장(built-in) 데이터 타입에 대해서 데이터베이스가 지원하는 모든 기능들, 즉 타입 시스템, 질의 처리 엔진, 인덱스 메커니즘 등을 사용자 정의 타입에 대해서도 똑같이 지원하는 것을 목표로 하고 있다. 이와 같은 확장성 기능을 IBM DB에서는 Data Extender[10], Informix(지금은 IBM에 흡수 합병되었음)에서는 Data Blade[11]라는 이름으로 제공하고 있다.

현재 ORDBMS 벤더들은 이 확장성 기능을 기반으로 새로운 데이터베이스 응용 분야를 위한 기능들을 구현해서 제공하고 있다. 오라클의 경우 자사의 확장성 기능[12]을 활용해서, 공간 데이터[13], 이미지[14], 텍스트[15], e-Commerce 데이터[16], 정규식(regular expression)[17] 등의 새로운 데이터 타입들을 확장성 기능을 기반으로 구현해서 제공하고 있다.

우선 확장형 인덱스의 개념을 쉽게 이해하도록 하기 위해서 정수형에 대한 B+ 트리 인덱스의 사용에 대해 생각해 보자. 우선 사용자는 임의의 테이블의 정수형에 대해 B+ 트리 인덱스를 생성할 수 있다. 그리고, 인덱스가 생성된 테이블에 새로운 레코드가 삽입/삭제/변경되면 마찬가지로 B+ 트리 인덱스의 내용도 따라서 변경된다. 그리고, 해당 테이블에 해당 칼럼에 조건(즉, 연산자 >, =, != 등)을 명시한 질의가 들어 왔을때, 해당 인덱스를 이용해서 조건을 만족하는 레코드들을 빨리 찾아내는 역할을 한다. 이들 기능들이 인덱스와 관련해서 사용자가 기대하는 부분들이다.

마찬가지로 확장형 인덱스는 사용자 입장에서 ① 사용자 정의 타입과 연산자를 위한 인덱스의 생성, ② 인덱스가 정의된 테이블에 레코드의 삽입/삭제/갱신이 발생할 경우 인덱스의 구조 변경 ③ 질의 처리시 특정 연산자를 인덱스를 이용한 빠른 처리를 제공해야 한다. (그림 4)는 이와 같은 확장형 인덱스의 개념을 잘 보여주고 있다.



(그림 4) 오라클 DBMS에서 확장성을 위한 인터페이스

(그림 4)의 왼편에 나와 있는 것처럼, 일반 사용자 입장에서는 사용자 정의 타입에 대해 인덱스 생성, 테이블에 레코드 삽입/삭제/변경을 수행한다. 이 경우 오라클 서버는 확장형 인덱스 개발자가 해당 인덱스에 대해 정의한 함수들 ((그림 4)에서는 ODCIxxxx로 표시된 함수들)을 내부적으로 호출해서 대응하는 일들을 수행한다. 실제 인덱스의 물리적 구조는 데이터베이스 내에 테이블의 형태로 저장된다.

3. XML 포함질의를 위한 확장형 인덱스의 구현

이상에서 살펴 본 기본 개념들을 바탕으로 본 장에서는 XML 포함질의를 위한 확장형 인덱스의 구현을 설명한다. 좀 더 구체적으로는, 확장된 역색인 인덱스 기법을 오라클 ORDBMS의 확장성을 이용해서 하나의 확장형 인덱스로 구현하는 방안을 설명한다. 본 논문에서는 우리는 ① 모든 XML 문서는 테이블에 BLOB 또는 CLOB 등의 형태로 데이터베이스에 저장되고, ② SQL에서 Contains라는 연산자를 이용해서 이 문서들에 대한 포함질의를 표현해서 검색하고, ③ 확장형 인덱스는 2.2절에서 설명한 4개의 테이블들에 저장되는 것을 가정한다.

본 장은 다음과 같은 순서로 구성되어 있다. 우선 사용자가 SQL을 이용해서 포함질의를 표현하는 방법을 설명하고, 다음으로 포함질의 처리를 위한 확장형 인덱스의 구현 내용을 설명한 후, 사용자가 이 확장형 인덱스를 생성/사용하는 방법을 설명한다.

3.1 SQL에서 포함질의의 표현 방법

(그림 1)의 예제 문서와 같은 XML 문서들을 company라는 테이블의 document 칼럼에 BLOB 형태로 저장하고 간접포함질의 /companies//profile//'scanners'를 수행할 경우에는 다음과 같은 SQL문을 수행하면 된다.

```
Company(doc_id number, document BLOB) ;
```

예제 질의 Q1 :

```
SELECT doc_id FROM company
WHERE  contains(document, '/companies//profile//"scanners") =
true ;
```

위에서 contains라는 연산자는 document라는 XML 문서가 포함질의를 만족하면 true를, 그렇지 않으면 false를 리턴하는 역할을 한다. 본 논문에서 제안하는 확장형 인덱스가 없는 경우, 위 질의를 처리하기 위해서는 company에 있는 각각의 XML document에 대해 다음 함수 contains_func을 등록하고, contains 연산자는 이 함수를 호출해서 where 절의 만족 여부를 검사한다. 우리는 contains_func에 대해서는 더 이상 자세히 다루지 않겠다.

```
/* Contains_Func 함수 생성 */
create or replace function contains_func(
  document BLOB, condition VARCHAR2)
  return BOOLEAN as
begin
  // 주어진 document를 파싱해서 주어진 포함질의의
     만족 여부를 검사
end within_distance_func ;

/* Contains 연산자 생성 */
create or replace operator contains
binding (BLOB, VARCHAR2)
return boolean
using contains_func ;
```

이처럼 contains라는 연산자를 선언해 주어야 오라클 DBMS는 이를 연산자로 인식하고, 일반적인 내장 연산자 >, <, = 등과 같이 취급을 하게 된다. 위 선언에서 using contains_func은, 인덱스가 정의되어 있지 않은 경우에, 연산자 contains는 이 contains_func 함수를 호출해서 처리하도록 지시하는 것이다.

그러나, 본 논문에서 제안하는 확장형 인덱스를 document 애트리뷰트에 생성하는 경우, contains 연산자의 처리는 company 테이블의 모든 XML 문서를 검색할 필요 없이 해당 조건을 만족하는 document만 검색할 수 있게 된다.

3.2 확장형 인덱스의 구현

이 절에서는 위에서 선언된 contains라는 연산자를 위한 확장형 인덱스를 오라클 확장성 기능을 이용해서 어떤 절차를 밟아서 생성하게 되는지를 설명한다. 본 절에서는 지면상 구현의 자세한 코딩 내용을 설명하는 대신에 각 구현 모듈의 개략적인 역할에 대해 설명한다.

3.2.1 인덱스 타입 선언

다음 타입 선언은 BLOB 타입으로 저장된 XML 데이터에 대한 인덱스 타입인 xml_idxtype_im의 선언을 보여 주고 있다.

```
CREATE OR REPLACE TYPE xml_idxtype_im AS OBJECT
( STATIC FUNCTION ODCIGetInterfaces(…) ;
  STATIC FUNCTION ODCIIndexCreate(…) ;
  STATIC FUNCTION ODCIIndexDrop(…) ;
  STATIC FUNCTION ODCIIndexStart(…) ;
  MEMBER FUNCTION ODCIIndexFetch(…) ;
  MEMBER FUNCTION ODCIIndexClose(…) ;
  STATIC FUNCTION ODCIIndexInsert(…) ;
  STATIC FUNCTION ODCIIndexDelete(…) ;
  STATIC FUNCTION ODCIIndexUpdate(…) ;
)
```

위 선언에서는 타입 xml_idxtype_im은 멤버 함수로써 모든 ODCI로 시작하는 함수명을 사용하고 있다. ODCI는 Oracle Data Catridge Interface로써, 오라클 엔진에서 확장형 인덱스에 대해 필요로 하는 함수들을 정의한 것이다. 즉, 실행시에 오라클 엔진에서는 확장형 인덱스에 대해 인덱스로써 필요한 연산들을 호출할 때 이 표준 인터페이스를 사용하게 된다. 따라서, 확장형 인덱스 개발자는 자신에 개발하는 인덱스 타입에 대해 정확하게 정의된 이들 인터페이스 함수를 정의해서 제공해야만 한다.

위의 선언에서 키워드 STATIC과 MEMBER는 객체지향 언어인 C++과 Java 등의 STATIC 및 MEMBER와 유사한 의미이다. 즉, 클래스 멤버 함수와 객체 멤버 함수와 유사한데, STATIC은 해당 클래스에 정의된 함수이고, MEMBER는 해당 클래스의 객체 타입들을 대상으로 수행되는 함수이다.

우선 ODCIGetInteface() 함수는 나중에 오라클 엔진에서 해당 인덱스 타입에 명시적으로 정의된 함수들이 어떤 것이 있는지를 리턴하는 함수이다.

다음에 나와 있는 함수들은 모든 인덱스에 대해 반드시 사용자가 정의해 주어야 하는 함수들로, 순서대로 인덱스 생성, 삭제, 조건을 주고서 조건에 맞는 레코드식별자(record identifier)를 찾기, 레코드 식별자를 하나씩 획득, 인덱스 검색 종료, 인덱스에 레코드 삽입, 삭제, 그리고 갱신에 해당하는 함수들이다. 실제로 내장형(built-in)으로 제공되는 인덱스 타입들에 대해서도 오라클 엔진과 인덱스 모듈 사이에는 이와 유사한 인터페이스를 가진다.

3.2.2 인덱스 타입 본체 생성

그럼 다음으로 위에서 선언한 인덱스 타입의 각각의 멤버 함수들을 어떻게 구현하는지를 예를 통해서 알아보자. 이는 다음에 나와 있는 것처럼, 인덱스 타입 본체(Index Type Body) 선언으로 구현한다. 이 절에서는 대표적인 멤버 함수인 ODCIIndexCreate, ODCIIndexStart에 대해 구체적으로 설명을 하고, 나머지 멤버 함수들에 대해서는 개략적인 동작원리만을 설명하면 구현 방법은 쉽게 이해할 수

있을 것이다.

인덱스 타입 본체의 정의는 다음과 같이 할 수 있다. IS와 END 사이에 위에서 선언한 함수들의 정의를 내장시키면 된다.

```
CREATE OR REPLACE TYPE BODY xml_idxtype_im
IS
 ..... -- 각각의 멤버함수들의 정의 포함
 .....
END ;
```

다음은 위 타입 본체의 정의에 포함될 ODCIIndexCreate 함수의 내용을 보여준다. 이 함수에서는 BLOB 타입의 XML 데이터에 대해 2.2절에서 설명한 확장 역인덱스 테이블들을 생성하는 역할을 한다. 그리고, stmt2 이후에서는 이렇게 생성된 테이블 구조에다 인덱스 대상 테이블에 이미 데이터가 존재하는 경우, 그 데이터들의 인덱스 엔트리를 생성/삽입하는 문장을 생성하고, 그 이후에 이 문장을 이용해서 데이터를 삽입하는 과정을 보이고 있다(B+트리의 경우에도 이미 인덱스가 만들어지는 테이블에 데이터가 들어가 있는 경우 해당 데이터들을 대상으로 인덱스를 구축하는 것과 마찬가지이다).

```
STATIC FUNCTION ODCIIndexCreate (··)
    RETURN NUMBER
is
··· // 변수 선언
BEGIN
    // Path 테이블 생성을 위한 SQL문 작성 및 실행
    stmt1 : = 'CREATE TABLE' ||
            ia.IndexSchema || '.' || ia.IndexName || '_path' ||
            '( r ROWID, path VARCHAR2(1000), pathID
            NUMBER)' ;
    cnum1 : = dbms_sql.open_cursor ;
    dbms_sql.parse(cnum1, stmt1, dbms_sql.native) ;
    junk : = dbms_sql.execute(cnum1) ;
    dbms_sql.close_cursor(cnum1) ;
    ···// 반복적으로 Term, PathIndex, TermIndex 테이블들을
    생성

    // 테이블에 기존 데이터가 있는 경우, 이를 위한 인덱스 데
    이터 생성
    stmt2 : =' SELECT P.ROWID, ia.IndexCols(1).ColName' ||
            ' FROM ' || ia.IndexCols(1).TableSchema || '.' ||
                    ia.IndexCols(1).TableName || 'P' ;
    cnum2 : = dbms_sql.open_cursor ;
    dbms_sql.parse(cnum2, stmt2, dbms_sql.native) ;
    junk : = dbms_sql.execute(cnum2) ;
    ···// 각 BLOB의 XML 문서에 대해 java script로 된 parser
        를 호출해서
    // 인덱스용 테이블에 필요한 데이터들을 임시 파일에 생성
       해서
    // 이를 인덱스용 테이블의 loader를 이용해서 적재
dbms_sql.close_cursor(cnum2) ;
    RETURN ODCICONST.SUCCESS ;
END ;
```

다음으로, 위에서 정의/생성된 인덱스 구조에 조건을 주어서 이를 만족하는 레코드식별자(rid)들을 검색하는 ODCIIndexStart에 대해 알아보자. B+ 트리도 추상적으로는 특정 검색 조건을 인자로 해서, 이 조건을 만족하는 모든 레코드식별자들을 찾아주는 것임을 상기하기 바란다. 다음 ODCIIndexStart()함수는 이를 위한 개략적인 단계를 보이고 있다. 우선 포함질의의 조건을 파싱해서 4가지 타입 중 어떤 유형인지를 파악하고, 각 타입에 따라 2.2절에서 언급한 [6]에서 제안한 방식으로 인덱스용 테이블들에 대한 SQL 문을 수행해서 해당 조건을 만족하는 XML 문서에 대한 RID를 찾아낸다.

```
STATIC FUNCTION ODCIIndexStart(···)
RETURN NUMBER is
    ··· -- 변수 선언
BEGIN
    ..... -- 오류 check 로직
    // 인덱스의 탐색키로 주어진 포함질의 조건에 대해 파싱을
       수행한다.
    // 각각의 포함질의의 종류에 따라 인덱스 테이블들에 대한
       SQL 문 생성
    // 예를 들어, 예제 질의 Q1에 대해서는 아래의 SQL문을 수행
       SELECT distinct TI.docID
       FROM  Path P, Term T, TermIndex TI
       WHERE P.path like '/companies%/profile%' and
       T.term ='scanners'
       and TI.pathID=P.pathID and TI.termID=T.termID
    ...
END ;
```

주목할 점은 본 논문에서 구현하고 있는 확장 인덱스는 XML 문서를 위한 contains라는 연산자만을 지원하는 인덱스 타입의 예를 보이고 있다. 만일 XML 문서들에 대해 contains 이외에 다양한 연산자들이 필요하다면, 위 ODCIIndexStart 함수에서 이들 연산자들의 이름을 파싱해서 각각의 연산자에 맞는 시맨틱으로 구현해서 레코드식별자들의 집합을 리턴해 주어야 한다. B+ 트리 하나를 이용해서 >, =, >= 등의 다양한 연산자를 지원하는 것과 마찬가지이다.

이렇게 해서 해당 조건을 만족하는 XML 문서들에 대한 레코드들을 검색한 후에는, ODCIIndexFetch를 이용해서 각 레코드식별자를 차례로 획득한 다음, ODCIIndexClose를 수행한다.

다음으로 인덱스 엔트리의 삽입, 삭제, 갱신에 대해 알아보자. ODCIIndexInsert는 ODCIIndexCreate에서처럼, 새로 삽입되는 XML 문서에 대해 파싱을 수행해서 인덱스 테이블들에 필요한 데이터를 생성한 후에 loader기능을 이용해서 삽입한다. ODCIIndexDelete는 주어진 레코드식별자에 해당하는 정보들을 인덱스 테이블들에서 모두 삭제한다. ODCIIndexUpdate는 삭제 후 삽입(Delete-then-insert) 시맨틱을 이용해서 구현했다. 부분적인 변경에 대한 좀 더 효

과적인 갱신 방안은 추후 연구 과제이다.

마지막으로, ODCIIndexDrop의 경우 모든 인덱스 테이블들을 삭제하면 된다.

3.2.3 인덱스 타입 생성

이제 마지막으로 위에서 정의한 xml_idxtype_im 타입을 이용해서 인덱스 타입 xml_idxtype을 선언하면 XML 문서의 contains 연산자를 위한 확장형 인덱스의 정의가 끝나게 된다. 다음은 이 예를 보이고 있다. 즉, 인덱스 타입 xml_idxtype은 contains 연산자를 위한 확장형 인덱스 타입으로써 이는 실제 xml_idxtype_im을 이용해서 구현됨을 선언하고 있다.

```
CREATE OR REPLACE INDEXTYPE xml_idxtype
FOR
   Contains(BLOB, VARCHAR2)
USING xml_idxtype_im ;
```

인덱스 타입의 본체 구현은 PL/SQL 이외에 Java, C 등의 언어로도 구현 가능한데, 성능이 중요한 경우에는 C 언어를 사용할 수도 있다.

4. 확장형 인덱스 생성 및 사용

3.2절의 내용은 주로 확장형 인덱스 기능을 이용해서 고급 개발자들이 수행하는 내용이고, 일반 개발자 및 사용자들은 이렇게 개발된 인덱스 타입을 마치 일반 인덱스인 B+ 트리나 비트맵 인덱스를 사용하듯이 사용하면 된다.

4.1 확장형 인덱스 생성

다음은 위에서 정의한 xml_idxtype을 이용해서 company의 document 필드에 인덱스를 생성하는 예를 보이고 있다.

```
create index test_xml_idxon company(document)
indextype is xml_idxtype ;
```

이 명령이 수행되면, 내부적으로는 앞에서 정의한 ODCIIndexCreate가 동작해서 인덱스 엔트리 저장을 위한 테이블들이 생성되고, 이미 company에 들어있는 XML 문서들에 대한 인덱스 엔트리를 생성해서 저장하게 된다. 따라서, 위의 예에 대해서는 다음 4개의 테이블이 자동적으로 생성된다. 즉, 사용자가 생성하는 하나의 논리적인 인덱스에 대해 내부적으로는 4개의 인덱스용 테이블이 생성된다. 인덱스의 이름에서 끝부분은 ODCIIndexCreate에서 자동적으로 붙인다. 주의할 점은 사용자가 생성하는 각각의 xml_idxtype 인덱스 마다 아래의 테이블 4개가 별도록 생성된다는 점이다.

```
test_xml_idx_path
test_xml_idx_term
test_xml_idx_pathidx
test_xml_idx_termidx
```

그리고, 이미 company 테이블에 데이터가 존재하는 경우, ODCIIndexCreate의 후반부 로직에 의해, company 테이블의 각 튜플들의 document 필드에 해당하는 XML 문서를 파싱해서 이를 위의 인덱스용 테이블들에 적재함으로써 인덱스의 생성이 완료된다.

4.2 확장형 인덱스를 이용한 질의 처리 과정

4.1절에서 처럼, 확장형 인덱스가 생성된 후에, 3.1절에서 예로 든 간접포함 질의 Q1이 company 테이블을 대상으로 수행된다고 가정하자.

예제 질의 Q1 :

```
SELECT doc_id FROM company
WHERE  contains(document, '/companies//profile//'scanners'')
= true ;
```

이 경우에는 오라클 서버는 contains라는 연산자를 위해 이용가능한 확장형 인덱스가 존재하는 것을 알기 때문에, ODCIIndexStart/Fetch/Close라는 함수를 호출함으로써 확장형 인덱스를 이용해서 질의를 처리하게 된다((그림 4) 참조).

우선 ODCIIndexStart에서는 위 질의의 contains 조건을 파싱해서 아래와 같은 인덱스용 테이블에 대한 SQL 문으로 변환을 수행한다.

ODCIIndexStart함수 내에서 수행되는 SQL 문 :

```
SELECT distinct TI.docID
FROM  Path P, Term T, TermIndex TI
WHERE P.path like '/companies%/profile%' and T.term = 'scanners'
and TI.pathID=P.pathID and TI.termID = T.termID
```

위 SQL문은 company 테이블의 튜플들 중에서 contains 조건식을 만족하는 튜플들의 레코드식별자들을 찾아내게 된다. 이후 ODCIIndexFetch에서는 이들 튜플을 하나씩 하나씩 fetch하는 역할을 하고, 이 모든 튜플들에 대한 처리가 끝나면 ODCIIndexClose가 호출된다.

4.3 확장형 인덱스의 유지관리

이후에, company 테이블에 XML 문서들을 삽입, 삭제, 또는 갱신하게 되면, 3.2.2절에서 설명한 바와 같이 ODCIIndexInsert/Delete/Update가 내부적으로 불려서 테이블의 데이터와 인덱스의 엔트리 사이의 일관성을 유지하게 된다.

5. 결론 및 향후 연구 과제

본 글에서는 현재의 ORDBMS에서 제공하는 확장성 기능을 이용해서 XML 문서들에 대한 포함질의를 효과적으로 지원하는 확장성 인덱스의 구현방안을 제시했다. 본 연구에서 제안한 확장형 인덱스를 통해서 XML 포함질의에 대한 효율적인 처리가 가능하고, 사용자의 입장에서는 인덱스의 생성과 질의 변환가 같은 부담에서 벗어나게 된다.

향후에는 본 연구에서 구현한 인덱스와 상용 ORDBMS에서 제공하는 포함질의 기능간의 성능 비교를 수행할 것이다. 또한, 구현 과정에서 발견된 인덱스 검색 과정에서 인덱스용 테이블들에 대한 SQL 수행시, 오라클 DBMS의 비용 예측의 한계[16]를 극복할 수 있는 방안이 필요하다. 마지막으로, 본 논문에서 제시한 방안을 확장해서 포함질의 이외의 기타 다른 유형의 XML 질의에 대해서도 지원하는 방안을 계속적으로 연구할 것이다.

참 고 문 헌

[1] T. Bray, J. Paoli, and C. Sperberg-McQueen, "Extensible markup language(XML)1.0," Technical report, W3C Recommendation, 1998.

[2] Don Chamberlin, "XQuery : An XML Query Language," IBM System, Vol.41, No.4, pp.597-615, Journal, 2002.

[3] Jayavel Schanmugasundaram, He Gang, Kristin Tuffe, Chun Zhang, David DeWitt, and Jeffrey Naughton, "Relational database for quering XML documents : limitationand opportunities," VLDB, 1999.

[4] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, and G. Lohman, "On supporting containment queries in relational database management systems," Proceedings of ACM SIGMOD, 2001.

[5] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, "Moderm Information Retrieval," Addison-Wesley Longman Inc., 1999.

[6] Chi-young Seo, Sang-Won Lee, Hyoung-Joo Kim, "An Efficient Inverted Index Technique for XML Documents using RDBMS," Information and Software Technology, Vol.45, No.1, pp.11-22, 2003.

[7] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, Shunsuke Uemura, "XRel : a path-based approach to storage and retrieval of XML documents using relational databases," ACM Transaction on Internet Technology, Vol.1, No.1, pp.110-141, 2001.

[8] Michael Stonebraker, "Inclusion of New Types in Relational Database systems," Proceedings of ICDE, 1986.

[9] Oracle Corp., "Oracle9i Data Cartridge Developer's Guide Release 2 (9.2)," http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/appdev.920/a96595/toc.htm.

[10] S. Debloch et al., "Extensible Indexing Support in DB2 Universal Database," Components Database System(Book Chapter), Morgan Kaufmann, 2001.

[11] R. Bliujute et al., "Developing a Database for a New Index," Proceedings of ICDE, 1999.

[12] J. Srinivasan et al., "Extensible Indexing : A Framework for Integrating Domain-Specific Indexing Scheme into Oracle8i," Proceedings of ICDE, 2000.

[13] R. K. V. Kothuri et al., "Quadtree and R-Tree Indexes in Oracle Spatial : A Comparison using GIS Data," Proceedings of ACM SIGMOD, 2003.

[14] Oracle Corp., "Oracle interMedia User's Guide and Reference (Release 9.0.1)," http://otn.oracle.com/docs/products/oracle9i/doc_library/release2/appdev.920/a88786/toc.htm.

[15] M. Annamalai, "Indexing Images in Oracle8i," Proceedings of VLDB, 2000.

[16] S. DeFazio et al., "The Importance of Extensible Database Systems for e-Commerce," Proceedings of ICDE, 2001.

[17] A. Yalamanchi et al., "Managing Expressions as Data in Relational Database Systems," Conference on Innovative Data Systems Research (CIDR), 2003.

[18] Patricia Selingeret al., "Access Path Selection in a Relational Database System," Proceedings of SIGMOD, 1979.

이 상 원

e-mail : wonlee@ece.skku.ac.kr

1991년 서울대 컴퓨터학과(학사)

1994년 서울대 컴퓨터학과(석사)

1999년 서울대 컴퓨터학과(박사)

1999년~2001년 한국오라클

2001년~2002년 이화여대 BK21 계약교수

2002년~현재 성균관대학교 정보통신공학부 전임강사

관심분야 : XML, 객체관계형 DB, DB튜닝, Data Warehouse, XML