객체 관계형 DBMS를 이용한 XBench 성능평가

김 재 욱*, 송 용 호*, 이 상 원*

Performance Evaluation of XBench using an Object-Relational DBMS

Jae-Uk Kim, Yong-Ho Song, Sang-Won Lee

요 약

XML은 데이터 표현과 교환을 위한 표준으로 급격히 자리잡아가고 있으며, XML 문서는 다양한 응용분야에 도입되고 있다. 이러한 흐름에 따라 데이터베이스 분야에서도 XML 문서 데이터의 효율적인 저장과 검색에 관한 연구가 활발히 진행되어왔다. 최근에 여러 상용 (객체) 관계형 DBMS에서 XML 데이터베이스를 지원하고 있으며, Native XML DBMS도 학계와 산업계에서 지속적으로 개발되고 있는 실정이다. 또한, 이러한 여러 종류의 XML 데이터베이스의 성능을 평가하기 위한 다양한 종류의 벤치마크가 제안되었다. 본 논문에서는 특정 객체 관계형 DBMS를 이용해서 XBench 벤치마크를 수행해서 객체 관계형 데이터베이스의 성능을 평가/분석하는데 향후 개선 방향을 밝히는데 그 목적이 있다.

Key Words : XML; XBench; 객체 관계형 DB; 성능 평가

ABSTRACT

XML is rapidly spreading as a standard for data representation and exchange, and XML documents are adopted in various applications. According to this trend, researches in database has also focused on efficient storage and retrieval of XML documents. Recently, major (object) relational DBMS vendors support XML functionality, and several native XML DBMSs have been developed in academic or industry side. In addition, various benchmarks have been proposed so as to evaluate these XML database performance. In this paper, we evaluate the XBench benchmark using a commercial object-relational DBMS, analyze its performance, and investigate the future improvements of object-relational DBMSs for XML support.

Ⅰ. 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 대한정보를 나타내기 위해 사용될 수 있다. 또한 전자상거래 사이트에서 각종 상품에 대한 정보를 나타내기 위해 사용될 수 있다. 결국, 이러한 추세로 방대한 생성되는 XML 문서들을 데이터베이스에서 효율적으로 저장, 조작, 검색을 하기 위한 연구가 필요하다.

이와 관련한 대표적인 접근 방법으로는 Flat file 시스템 (Kweelt[3]), 관계형 DBMS(Oracle, IBM DB2, Microsoft SQL Server)나 객체지향 DBMS(Ozone[4]), Native XML Repository(Tamino[5], Natic[6], X-Hive, Xyleme)등의 시스템에서 여러 접근 방식들이 제안되었다. 그러나, 주요 DBMS 벤더들이 XML 기능을 지

* 성균관대학교 정보통신공학부 VLDB 연구실({wodnr,neilkr,swlee}@skku.edu)
논문번호 : KICS2004-10-248, 접수일자 : 2004년 10월 25일
※이 논문은 성균관대학교의 2003학년도 성균학술연구비에 의해 연구되었음

원함에 따라 궁극적으로는 상용 관계형 DBMS 또는 이들이 진화한 객체 관계형 DBMS가 주류 XML 데이터베이스 서버로 자리 잡게 될 것이다. 특히, XML 전용 서버로 개발된 제품들의 경우, DBMS로써의 핵심적인 트랜잭션 기능과 질의 최적화 등의 핵심적인 모듈 부분에서 주요 상용 DBMS가 20년 이상 축적한 기술 장벽을 극복하기 쉽지 않을 것이다. 이러한 가정에 기반으로 본 논문에서는 향후 주류 XML 데이터 저장고 역할을 하게 될 객체 관계형 DBMS를 이용해서 대표적인 XML 벤치마크를 순수 관계형 모델링과 객체 관계형 모델링 기법으로 저장해서 두 접근 방법 사이의 성능을 비교분석하고, 객체 관계형 접근방법에서 실무적인 측면에서 성능 개선 방안을 도출하고자 한다.

본 연구의 기여사항은 다음과 같다. 1) 대표적인 XML 벤치마크를 순수 관계형 모델 및 객체 관계형 기능을 사용해서 모델링하고 데이터를 구축하고, 2) XQuery를 이들 모델링 기반의 SQL 질의로 변환하고, 3) 두 접근 방법사이의 성능을 비교하고, 4) 객체 관계형 접근 방법에서 다양한 성능 개선을 위한 방안을 제시한다.

본 논문은 구성은 다음과 같다. 2장에서는 본 논문의 실험에 사용된 XBench 벤치마크의 내용을 소개하고, 3장에서는 본 논문의 실험에 사용된 객체 관계형 DBMS(이하 A사의 객체 관계형 DBMS라 칭함)에서 XML 지원 방안을 설명하고, XBench 스키마를 순수 관계형 모델과 객체 관계형 모델을 사용해서 스키마를 설계한 내용을 설명한다. 4장에서는 이 두 가지 모델을 사용한 성능 실험에 대해 서술하고, 5장에서는 실험을 통해 얻은 결과를 바탕으로 성능을 평가하고 분석한 것을 서술한다. 마지막으로 6장에서는 결론을 기술한다.

Ⅱ. XBench 벤치마크 소개

현재까지 XML용 벤치마크로는 XBench[8], XMach-1[10], Xmark[11], XOO7[12], Micro Benchmarks[13]등 다양한 종류가 제시되었다. XBench 외의 벤치마크들은 단순한 스키마와 저용량의 XML 데이터에 대해 벤치마크를 수행하므로, 본 논문에서는 다양한 스키마와 대용량 데이터를 기반으로 실험이 가능한 XBench 벤치마크를 선택해서 순수 관계형 모델링과 객체 관계형 모델링 방법의 성능을 비교한다.

XBench 벤치마크에서 사용되는 데이터는 small(10M), normal(100M), large(1GB), huge(10GB)의 테스트 환경이 있으며, 다음의 두 가지 특성으로 구분하고 있다. 첫 번째 특성은 애플리케이션 특성이고, 두 번째 특성은 데이터 특성으로 구분하고 있다[8,15]. 좀 더 구체적으로, 애플리케이션 특성에는 데이터 중심(Data-centric, DC)으로 된 XML 문서와 텍스트 중심(Text-centric, TC)으로 된 XML 문서로 구분한다. 데이터 중심의 XML 데이터는 전자상거래 카탈로그 정보나 XML 데이터로 입력된 업무 처리데이터를 이용한 정보들을 말한다. 텍스트 중심의 XML 데이터는 XML 문서로 만들어진 실제 텍스트 데이터를 다룬다. 예를 들면 뉴스 기사나 디지털 도서관의 책 내용들, 그리고 사전 등이 이에 해당한다. 데이터 특성은 두 가지 항목으로 구분 할 수 있다. 하나의 XML 파일로 이루어진 단일 문서(Single document, SD)와 여러 개의 XML 파일로 이루어진 다수 문서(Multiple document, MD)로 구분된다. 단일 XML 문서는 전자상거래 카탈로그 데이터나 XML 데이터로 입력된 업무 자료를 이용한 복잡한 구조로 설계된 데이터에 대해 적용한 것이다. 다수의 XML 문서는 뉴스 기사나 업무처리 자료를 기록한 XML 문서를 데이터베이스에 적용한 것이다. 따라서 데이터베이스에 적용할 수 있는 XML 데이터는 <표 1>과 같이 네 가지 항목에 대한 XML 데이터를 적용할 수 있다.

표 1. XML 데이터 특성

	SD	MD
TC	전자 도서관	뉴스 기사, 전자 도서관
DC	전자상거래 카탈로그	업무처리 데이터

본 논문에서는 데이터 중심(DC) XML 데이터를 A사 객체 관계형 DBMS를 이용해서 순수 관계형과 객체 관계형으로 모델링해서 성능을 비교 평가한다. 텍스트 중심(TC) XML 데이터의 경우, 대량의 데이터 적재, CLOB 타입 등 순수 관계형 및 객체 관계형 기능을 이용해서 구현하는 과정에서 해결되지 못한 문제 등으로 향후 연구과제로 남겨둔다.

데이터 중심 XML 문서는 업무를 처리할 때와 문서 교환에 주로 사용될 수 있는 구조를 가지고 있다. DC/SD 문서는 단일 XML 문서로 되어있으며, 스키마는catalog를 최상위 노드로 하는 <그림 1>과 같은 구조를 가진다. DC/MD 문서는 다수의 XML 문서로 구성되어 있고 각 문서의 크기는 작다(수 K 바이트 이내). DC/MD의 스키마는 <그림 2> ~ <그림 7>에서와 같은 구조를 가지고 있다. XBench 벤치마크의 자세한 스키마에 대한 정보는 참고문헌[8]을 참조하기

바란다. 모든 객체 관계형 데이터베이스와 관계형 데이터베이스에 대한 실험은 large(1GB) 크기의 XML 데이터로 실험한다. small(10MB)과 normal(100MB) 데이터들은 실험 결과 두 방법사이에 별 차이가 없어 성능 비교의 의미가 별로 없는 것으로 판단된다. huge(10GB)의 경우 (본 저자들의 상용 DBMS 사용 방법에 대한 지식의 한계 또는 해당 DBMS의 기능의 한계인지는 몰라도) 객체 관계형 모델링에 데이터를 적재하는 과정에서 과다한 시간이 소요되었기 때문에 실험을 수행하지 못했다.

Ⅲ. A사의 객체 관계형 DBMS에서 XML 지원

A사의 상용 객체 관계형 DBMS에서는 객체 관계형 기능들을 활용해서 XML 문서의 저장과 검색 기능을 제공하고 있다. 또한 W3C XML 데이터 모델을 완벽하게 수용하고 있으며, 이에 따른 XML Repository의 기능과 XPath와 기존의 순수 관계형 데이터베이스의 SQL 질의를 사용하여 데이터에 대한 표준 접근 방식을 사용할 수 있도록 하고 있다. 다음에 기술된 내용들은 XML 스키마를 어떻게 객체 관계형 데이터베이스에 모델링 하는지를 보여준다.

XML 스키마는 XML 문서를 데이터베이스에 삽입하기 전에 해당 스키마 정보를 미리 등록해야 한다. XML 스키마를 등록하게 되면 등록과 동시에, 등록된 XML 스키마 정의를 기반으로 DBMS에서 자동적으로 필요한 데이터베이스 객체들 - 즉 테이블, 내부 테이블(nested table), 콜렉션 타입(collection type), 오브젝트 타입(object type) 등의 객체 관계형 모델에서 지원하는 객체들, - 을 생성하게 된다. <그림 8>은 DC/MD 스키마 중 <그림 3>의 DCMDAddr.xsd 스키마를 등록했을 때 A사 DBMS가 데이터베이스 객체를 생성하는 예를 보여주고 있다.

addresses를 최상위 요소로 하고 있는 스키마 구조를 가지고 있으며, address와 street_address 는 VARRAY(Variable Array) 형태로 설계되어 있는 것을 볼 수 있다. 이 정의는 XML 스키마에서 그 형태를 Ordered Collection Table(OCT) 형태로 저장되도록 정의해 주는 것과 같다. 다시 말하자면, XML 스키마를 등록하면서 VARRAY 형태로 정의되어 있는 요소들에 대해 Indexed Organized Table(IOT) 테이블로 생성하여 찾고자 하는 데이터에 대한 접근을 최적화된 형태로 저장할 수 있도록 정의하는 것이다. 이렇게 생성된 IOT 테이블은 시스템에서 정의되어 있기 때문에 사용자가 접근할 수 없는 형태로 정의되며,

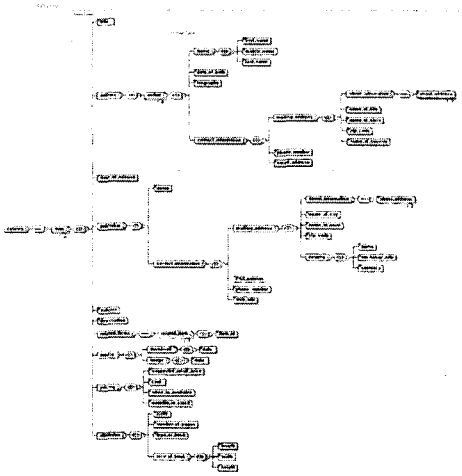


그림 1. DC/SD의 catalog 스키마

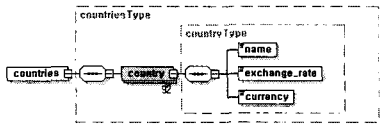


그림 2. DC/MD의 countries 스키마

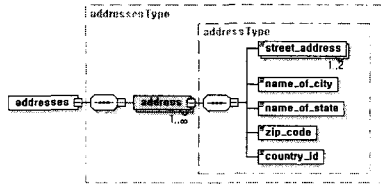


그림 3. DC/MD의 addresses 스키마

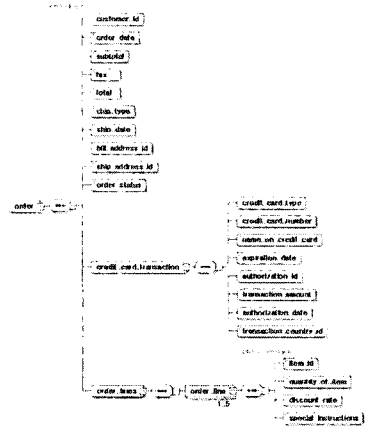


그림 4. DC/MD의 order 스키마

데이터 검색과 추출과정에서 최적화기의 판단에 따라 적절히 사용되게 된다.<그림 8>은 또 다른 예로써 DCMDAddr.xsd 스키마가 객체 관계형으로 어떻게 모델링 되는지를 보여주고 있다. 한편, <그림 9>는 DCMDAddr.xsd를 순수 관계형 데이터베이스로 설계한 내용을 보이는데, 이 모델링 방법은 참고문헌[8]에서 사용된 스키마를 그대로 활용했다.

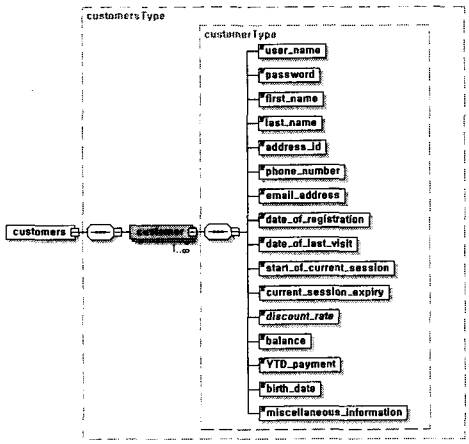


그림 5. DC/MD의 customers 스키마

Ⅳ. 성능 평가 및 분석

4.1 실험 환경 및 XQuery 표현

본 절에서는 A사의 DBMS를 이용해 XBench 벤치마크의 성능 테스트를 수행한 실험환경과 XQuery 질의를 표현한 방법을 기술한다. 실험을 위해 사용된 시스템은 1GB 주 기억장치, 40GB 하드디스크, 팬티엄 4 2.0Ghz, RedHat Linux 3 AS 시스템에 A사 DBMS의 가장 최신 DBMS를 탑재하여 수행했다. 본 논문에서는 스키마 DC/MD와 DC/SD에 대해 large(1GB)의 데이터 크기에 대해 수행시간을 측정했다.

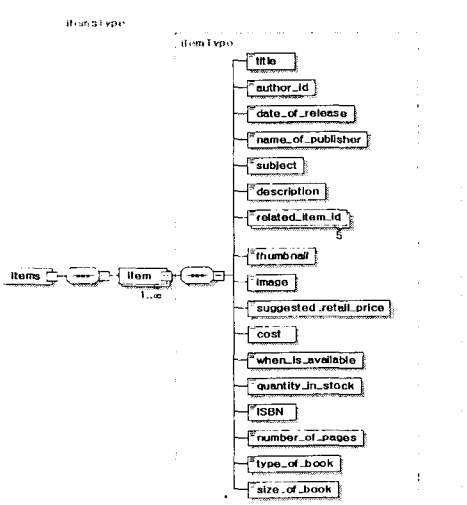


그림 6. DC/MD의 items 스키마

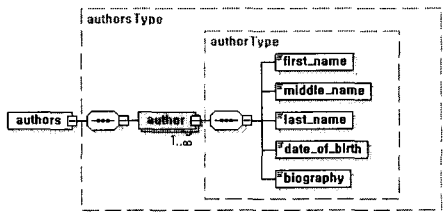


그림 7. DC/MD의 authors 스키마

XBench벤치마크에서 제공하는 XQuery 질의들은 각 스키마 별로 20개의 질의로 구성되어 있다. A사 데이터베이스에서는 XQuery를 직접 지원하지 않기 때문에 각 XQuery 질의를, 객체 관계형 기능을 사용한 경우에는 XPath 형태로, 순수 관계형 기능을 사용한 경우에는 SQL 질의 문으로 번역하여 실험하였다. XPath 형태의 질의는 실제로 질의 최적화기에 의해 자동적으로 객체 관계형 기능을 이용한 SQL 문으로 재작성(rewrite)되어서 수행되는데, <표 2>는 XPath가 SQL 형태로 변환되어 수행되는 질의 문의 예를 보여주고 있다. XBench 벤치마크에서 사용하는 XQuery 질의들에 관해서는 참고문헌[8]을 참고하기 바란다. 각각의 스키마에 적용된 인덱스는 <표 3>에 명시되어 있는 각 칼럼에 대해 생성하였다.

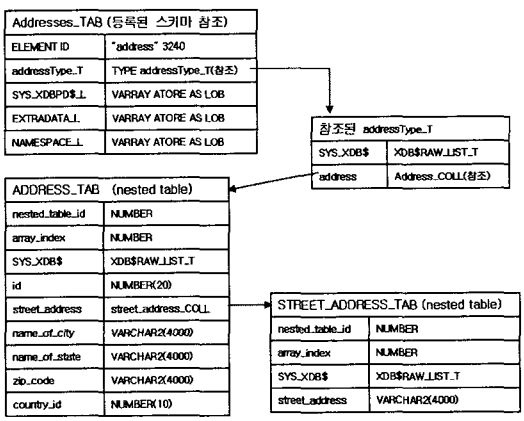


그림 8. DCMDAddr.xsd 스키마에 대한 객체 관계형 스키마 구조

표 2. XPath 질의가 SQL 질의로 번역되는 예

XPath 질의 (DC/MD 의 질의 1번)
SELECT extractValue(value(T1), '/order/customer_id') FROM order_tab T1 WHERE existsNode(value(T1), '/order[@id=1]') = 1;
XPath의 번역된 질의(DC/MD 의 질의 1번)
SELECT T1.XMLData."customer_id" FROM order_tab T1 WHERE T1.XMLData."id" = 1;

표 3. 생성되는 인덱스

	순수 관계형 데이터베이스	객체관계형 데이터베이스
DC/SD	date_of_release, first_name, name_of_country, description, publisher_name, item_id	item/@id, date_of_release, description, publisher_name, first_name, name_of_country
DC/MD	discount_rate, total, biography, customer_id	order/@id, customer/@id, total, discount_rate, biography

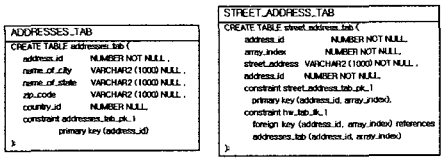


그림 9. 관계형 데이터베이스 생성

4.2. 성능 평가 및 분석

이번 장에서는 XBench 벤치마크의 각 질의에 대해 순수 관계형 모델링과 객체 관계형 모델링을 사용한 경우의 질의 수행결과를 설명하고, 성능 측면에서 두 방법을 비교한다. 모든 질의 시간측정은 각 질의마다 모든 데이터가 디스크 상에 있는 경우, 즉 버퍼에 데이터가 올라와 있지 않은 상태에서 수행하였다. <표 4>는 각각의 질의를 실행하여 얻은 결과를 나타낸다.

DC/SD와 DC/MD 데이터에 대한 성능은 <표 4>에서 보이는 바와 같이, 몇 개의 질의를 제외하고 거의 비슷한 수준의 성능을 보장하고 있다. 비슷한 성능을 보이는 질의의 경우는 순수 관계형과 객체 관계형을 이용해서 모델링했지만, 질의의 대상이 되는 데이터들의 스키마 구조가 거의 동일하기 때문이다. 따라서, 본 절에서는 성능 차이가 큰 질의들을 중심으로 성능 비교를 하겠다.

4.3. 성능 평가 및 분석

4.3.1. DC/MD

1) Q3 : 루트 노드인 order 의 하위 노드인 total 노드에 인덱스가 생성되어 있다. 이와 같은 경우 total의 값이 11000.0 보다 큰 값을 찾아 customer_id와

표 4. 각 스키마의 실험 결과

	DC/MD		DC/SD	
	X-DB	R-DB	X-DB	R-DB
Q1	0.05	0.04	0.08	0.19
Q2	N/A	N/A	1.52	1.76
Q3	4.87	21.05	10.60	4.81
Q4	0.15	0.05	51.57	1.29
Q5	0.11	0.06	0.10	0.11
Q6	125.21	25.71	57.73	133.94
Q7	점검	점검	점검	점검
Q8	0.18	0.04	0.08	0.13
Q9	0.24	0.04	0.08	0.07
Q10	4.91	21.44	68.59	11.04
Q11	4.78	20.19	68.73	10.71
Q12	0.18	0.06	0.15	0.14
Q13	N/A	N/A	N/A	N/A
Q14	657.06	9.14	9.52	22.66
Q15	N/A	N/A	N/A	N/A
Q16	0.06	0.05	N/A	N/A
Q17	5.50	1.01	54.79	9.21
Q18	N/A	N/A	N/A	N/A
Q19	4.45	0.12	0.13	0.13
Q20	N/A	N/A	589.93	4.48

개수를 customer_id로 GROUP BY하여 출력하는 질의인데, 객체 관계형 데이터베이스와 관계형 데이터베이스의 최적화기의 판단에 차이가 발생한다. 관계형 데이터베이스의 경우 total에 대한 인덱스를 사용하는 반면, 객체 관계형 데이터베이스는 전체 검색(full table scan)을 하는 실행계획을 사용한다. 이와 같은 경우 관계형 데이터베이스의 성능이 좋지 않은 이유는 클러스터링 팩터(clustering factor)[14]가 좋지 않은 것이 원인이므로 인덱스를 사용하면 나쁜 성능을 보이게 된다. 따라서 인덱스를 사용하지 않고 전체 검색을 하게 되면 객체 관계형 데이터베이스와 같은 성능을 나타내게 된다.

2) Q6 : 특정 노드의 값이 원하는 값 이상일 때 특정 노드의 값을 반환하는 질의이다. 이 경우 객체 관계형 데이터베이스의 성능이 좋지 않게 나타나고 있는데, 그 이유는 내부 테이블내의 특정 노드의 값을 비교하는 과정에서 비용이 상당히 많이 들게 된다. 또한 비교하여 얻은 결과는 루트 테이블의 nested_id 값과 조인이 발생하는데 여기에 드는 비용도 상당히 많다. 이에 비해 관계형 데이터베이스의 경우 두 테이블을 전체 검색을 하면서 만족하는 결과를 가져오므로 비용이 객체 관계형 데이터베이스에 비해 적게 든다.

3) Q10 : 이 질의 또한 Q3과 같은 원인 때문에 관계형 데이터베이스가 성능이 나쁘게 나타나고 있다.

total 노드의 값이 11000.0 보다 큰 값을 찾아 order의 id값과 order_date, ship_type을 ship_type으로 ORDER BY 하여 출력하는 질의이다. 이 질의 또한 객체 관계형 데이터베이스는 전체 검색을 하는 반면 관계형 데이터베이스는 클러스터링 팩터가 좋지 않은 상태에서 인덱스를 사용하기 때문에 나쁜 성능을 보이고 있다.

4) Q11 : 이 또한 Q3질의와 같은 맥락으로 클러스터링 팩터의 영향에 의해 관계형 데이터베이스가 좋지 않은 성능을 보이고 있다. total 노드의 값이 11000.0 보다 큰 값을 찾아 order의 id값과 order_date, total을 total로 ORDER BY 하여 출력하는 질의이다. 관계형 데이터베이스에서 total 칼럼의 인덱스를 사용하지 않으면 성능향상을 할 수 있다.

5) Q14 : 이번 질의는 특정 VARRAY 형태로 되어있는 노드에 대해 값이 NULL인 ARRAY 값을 찾아 특정 데이터 값을 출력하는 질의이다. 관계형 데이터베이스의 경우 order_line의 id 값을 count하여 그 값이 1인 것만 출력하는 단순한 처리과정을 거치기 때문에 빠른 응답 속도를 보인다. 반면 객체 관계형 데이터베이스의 경우는 전체 검색을 통해 array index 칼럼에서 존재여부를 판단하기 때문에 I/O발생이 굉장히 많아지므로 좋지 않은 성능을 보일 수 있다. 이 같은 경우 해당하는 VARRAY를 외부 테이블로 정의하여 정의된 테이블로 직접 접근 하면 어느 정도 성능향상을 기대할 수 있다.

6) Q17 : 이 질의 같은 경우 특정 노드에 포함된 텍스트 값들 중 원하는 값을 얼마나 신속히 찾을 수 있는지를 테스트 하는 질의이다. 관계형 데이터베이스에 비해 객체 관계형 데이터베이스가 좋지 않은 성능을 보이는 이유는 객체 관계형 데이터베이스의 특성상 조인이 한 번 더 발생하기 때문이다. 루트 노드인 authors의 하위에 있는 author의 하위 노드인 biography의 데이터 중 특정 값과 일치하는 값이 있으면 author의 id값을 출력하는 질의이다. 이 같은 스키마는 모델링하는 과정에서 루트 테이블이 있고, 루트 테이블의 한 칼럼에 데이터가 들어있는 테이블이 내부 테이블의 형태로 구성된다. 따라서 루트 테이블의 nested_id 값과 내부 테이블에서 일치하는 값에 대한 nested_id 값을 조인하여 찾아내어 실행하게 된다. 반면 관계형 데이터베이스 같은 경우 단순히 author 테이블을 전체 검색을 하여 부합하는 데이터가 있을 때 마다 출력해 주기 때문에 좋은 성능을 보인다. 따라서 객체 관계형 데이터베이스의 성능 향상을 기대하자면 단순히 스키마 구조를 변경해 주는 것으로 해결할 수 있다. 루트 노드인 authors의 하위 노드 author 노드를 외부 테이블로 정의해 주고 author테이블에 직접 접근하면 관계형 데이터베이스와 같은 성능을 보일 수 있다

7) Q19 : 이번 질의는 관계가 형성되어 있지 않은 두 스키마에서 서로 order의 id값이 일치하는 것을 찾아 원하는 노드의 값을 출력해 주는 질의이다. 이 경우 객체 관계형 데이터베이스가 나쁜 성능을 보이고 있는데, 원인은 관계형 데이터베이스에 비해 조인이 한 번 더 발생하여 비용이 더 많이 들기 때문이다. 이 또한 스키마 구조를 변경해 주는 것으로 관계형 데이터베이스만큼의 성능을 기대할 수 있다. 루트 노드인 customers 하위의 customer 노드에 대해 외부 테이블로 정의해 주고 customer 테이블로 직접 접근하여 데이터를 찾는다면 비슷한 성능을 보장할 수 있다.

4.3.2. DC/SD

1) Q3 : 관계형 데이터베이스에서는 전체 검색을 통해서만 데이터를 추출하는 반면, 객체 관계형 데이터베이스의 경우 내부 테이블에서 전체 검색을 통해 추출된 결과를 루트 테이블과 nested_id에 대한 hash 조인이 발생한다. 여기에서 비용이 많이 들기 때문에 관계형 데이터베이스 보다 나쁜 성능을 보이고 있다.

2) Q4 : 관계형 데이터베이스에서는 유일키를 통해 만족하는 데이터를 찾고 셀프 조인하여 원하는 결과를 검색한다. 반면 객체 관계형 데이터베이스에서는 각각의 내부 테이블에서 인덱스를 통해 만족하는 데이터를 찾고 셀프 조인을 한 후 루트 테이블과 nested_id를 비교하기 위해 한 번 더 조인을 하게 된다. 여기에서 드는 비용이 많기 때문에 나쁜 성능을 보이고 있다. 내부 테이블을 외부 테이블로 모델링 한다면 이러한 비용을 줄일 수 있다.

3) Q6 : 이 질의의 경우 객체 관계형 데이터베이스일 때 내부 테이블 형태로 정의된 노드 하위의 데이터에 대해 만족하는 값에 대한 특정 노드의 결과 셋을 출력하는 질의이다. 실험 결과는 내부 테이블일 때 결과 셋이며, 이것을 외부 테이블로 모델링하게 되면 더 좋은 성능을 나타낼 수 있다. 관계형 데이터베이스의 성능이 문제가 되는 것은 설계된 모든 테이블에 대한 조인이 발생하여 여기에 드는 비용이 상당히 많이 드는 것이 문제이다.

4) Q10 : 이 질의는 date_of_release의 값이 특정 날짜 사이에 있는 값에 대해 특정 노드의 값을 출력하는 질의이다. 이 경우 관계형 데이터베이스는 전체

검색을 통해 만족하는 값을 정렬하여 출력하는 반면 객체 관계형 데이터베이스는 내부 테이블이기 때문에 전체 검색 후 루트 테이블과 nested_id 값이 일치하는 값을 찾기 위해 hash 조인을 한 후 정렬하여 출력하기 때문에 관계형 데이터베이스에 비해 좋지 않은 성능을 보이고 있다. 객체 관계형 데이터베이스의 성능을 향상시키기 위해서는 VARRAY로 정의된 item 노드를 외부테이블로 정의하면 좋은 성능을 기대할 수 있다.

5) Q11 : 이 질의 또한 Q10 질의와 같은 원인으로 객체 관계형 데이터베이스의 성능이 나쁘게 나타나고 있다. 이 또한 외부 테이블로 정의해 주면 기대할 만한 성능을 보장할 수 있다.

6) Q14 : 이번 질의 또한 Q10, Q11과 거의 유사한 형태를 가지고 있는 질의이지만 최적화기의 잘못된 판단 때문에 관계형 데이터베이스의 성능이 나쁘게 나오고 있다. 이번 질의 같은 경우 date_of_release에 대해 인덱스를 사용하는 실행 계획이 나타나는데 나쁜 클러스터링 팩터의 원인으로 전체 검색을 하는 경우가 좋은 성능을 보이고 있다. 관계형 데이터베이스에서 인덱스를 사용하지 않으면 객체 관계형 데이터베이스와 동일한 성능을 보장할 수 있다.

7) Q17 : DC/MD의 Q17과 같이 노드에 포함된 텍스트 값들 중 원하는 값을 얼마나 신속히 찾을 수 있는지를 테스트 하는 질의이다. DC/MD와 같은 원인으로 객체 관계형 데이터베이스의 성능이 나쁘게 나오고 있다. 역시 원하는 특정 노드를 외부 테이블로 정의함으로써 성능향상을 기대할 수 있다.

8) Q20 : 이번 질의는 특정 노드의 하위에 있는 노드들의 값들을 곱하여 일정 수치 이상인 것을 가려내어 원하는 노드의 값을 출력하는 질의이다. 관계형 데이터베이스의 경우 하나의 테이블에 곱해야 할 값들이 모두 존재하여 각 칼럼의 값들을 곱하여 만족하는 값을 찾는 연산이므로, 신속하게 처리하는 반면, 객체 관계형 데이터베이스의 경우 각 값마다 각기 다른 테이블로 구성되어 각각의 값들을 찾을 때 내부 테이블의 nested_id 값을 일일이 비교하여 찾고, 그 값들을 곱하는 연산이 발생하게 된다. 따라서 관계형 데이터베이스가 비용이 적게 들기 때문에 좋은 성능을 보이고 있다. 객체 관계형 데이터베이스는 이러한 경우에 대해 스키마 최적화나 질의 최적화를 통해 성능향상에 대한 방법을 모색해야 한다.

V. 객체 관계형 모델에서 성능개선 방향

앞 절에서 설명한 실험에서 객체 관계형 모델을 이용한 경우에는, A사 DBMS에서 제공하는 기본 옵션을 사용해서 XSchema를 등록하면 자동적으로 만들어지는 객체 관계형 스키마에서는 모든 VARRAY가 중첩 테이블(nested table)로 생성된다. 이러한 이유로, 몇몇 질의에 대해 나쁜 성능을 보이는데, 이 절에서는 예를 통해서 중첩테이블을 사용하지 않고 외부 테이블로 정의했을 때 성능이 향상되는 것을 보이겠다.

<그림 10>은 DCMDOrd.xsd 스키마의 일부를 보인 것이다. 객체 관계형 데이터베이스에서 VARRAY로 정의된 order_line을 중첩 테이블과 외부 테이블로 정의한 두 개의 스키마에 대해 <표 5>의 질의를 실험한다. <표 5>에서 보이는 바와 같이 스키마 정의를 변경하여 객체 관계형 데이터베이스를 모델링 하는 것만으로도 그렇지 않은 경우에 비해 2배 정도의 성능 향상을 보이고 있다.

본 논문에서는 기본적인 XML 스키마 정의에 벗어나지 않는 범위에서 실험하였기 때문에 이와 같은 결과는 기대할 수 없다. 하지만 스키마 재 정의에 의한 객체 관계형 데이터베이스의 모델링을 한다면 순수 관계형 데이터베이스와 차이 없는 성능을 보일 수 있다고 기대된다.

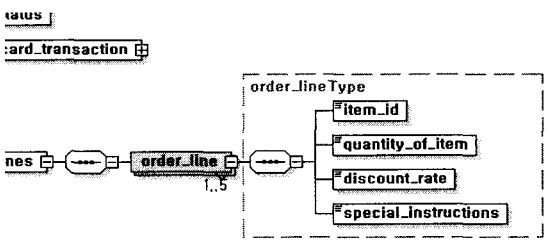


그림 10. DCMDOrd.xsd 스키마의 일부

표 5. 객체 관계형 DB에서의 성능 개선의 예

테스트 질의
내부 테이블의 경우
select extract(value(L), '/order_line/quantity_of_item') from order_tab 0, table(xmlsequence (extract(0.object_value,' /order/order_lines/order_line'))) L where existsNode(value(L), '/order_line[item_id=15089]') = 1;
결과 : 13.51
외부 테이블의 경우
select extract(value(0), '/order_line/quantity_of_item') from order_line_tab 0 where existsNode(value(0),, '/order_line[item_id=15089]') = 1;
결과 : 7.15

VI. 결론 및 향후 연구

본 논문에서는 XBench 벤치마크를 이용해서 객체 관계형 데이터베이스와 순수 관계형 데이터베이스의

기능에 대한 성능평가를 수행하였다. XBench 벤치마크를 통해 다양한 형태의 스키마와 데이터에 대한 포괄적인 테스트를 할 수 있었다. 순수 관계형 모델에 비해 객체 관계형 모델의 장점은 사용상의 편의성이라 할 수 있겠다. 모델링에서 어느 정도의 자동화된 기능, 단순하고 명확한 XPath의 질의 구조, 데이터의 저장 시 중복을 최소화할 수 있는 것은 객체 관계형 기능을 사용했을 경우의 큰 장점이라 할 수 있다. 또한 이러한 테스트를 통해 A사의 상용 객체 관계형 DBMS를 사용해서 객체 관계형 기능과 순수 관계형 기능을 모두를 테스트해봄으로써 두 방법의 장/단점을 비교할 수 있었다. 특히 객체 관계형 기능을 사용하는 경우, 성능 상 크게 세 가지 문제점이 발생한다. 우선, 질의 최적화기(query optimizer)가 상대적으로 질의 최적화를 제대로 수행하지 못한다. 또한, 중첩 테이블의 사용에 따른 질의 수행 성능의 저하를 들 수 있다. 셋째, XML 스키마를 객체 관계형 모델로 변환해주는 스키마 변환기에서 XML 스키마의 각 칼럼을 각각의 테이블로 나누어 모델링 하는 부분이 성능 저하에 주요한 이유가 된다. 따라서, 순수 관계형 기능을 사용한 경우에 비해서는 이러한 성능 병목 현상에 대한 해결책이 강구되어야 할 것이다.

향후 연구로는 다음 몇 가지를 들 수 있다. 우선, 객체 관계형 기능을 사용할 경우 XSchema에 다양한 힌트를 주어서 주어진 질의 부하에 대해 최선을 성능을 보일 수 있도록 annotation을 추가해서 최적의 객체 관계형 모델을 생성하는 방법론에 관한 연구가 필요하다. 또한, XML 측면에서 라기 보다 좀 더 일반적인 관점에서 객체 관계형 기능을 포함한 스키마와 질의들의 질의 수행에 대한 이해와 질의 최적화에 대한 일반적인 연구가 필요하다.

참 고 문 헌

[1] Intelligent Enterprise Corporation, "IDC Hails XML as B2B Integration Standard"

[2] Duckett, Jon, Professional XML Schemas, 2003.

[3] A. Sahuguet. KWEELT, the Making-of: Mistakes Made and Lessons Learned. Technical report, Department of Computer and Information Science, University of Pennsylvania, November 2000.

[4] T. Lahiri, S. Abiteboul, and J. Widom. Ozone: Integrating Structured and Semistructured Data. In Proc. 7th Int. Conf. on Database Programming Languages, pages 297-323, September 1999.

[5] H. Schoning and J. Wasch. Tamino An Internet Database System. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, Advances in Database Technology EDBT 2000, Proc. 6th Int. Conf. on Extending Database Technology, volume 1777 of Lecture Notes in Computer Science, pages 383-387. Springer, 2000.

[6] T. Fiebig, S. Helmer, C.-C. Kanne, J. Mildenberger, G. Moerkotte, R. Schiele, and T. Westmann. Anatomy of a Native XML Base Management System. Technical Report TR-02-001, Universitat Mannheim, January 2002.

[7] B. B. Yao, M. T. ozsu, and N. Khandelwal, "XBench Benchmark and Performance Testing of XML DBMSs", In Proceedings of 20th International Conference on Data Engineering, Boston, MA, March 2004, pages 621-632.

[8] Benjamin Bin Yao and M. Tamer ozsu, Evaluation of DBMSs Using XBench Benchmark, TR-CS-2003-24, University of Waterloo, August 2003.

[9] Ravi Murthy, Sandeepan Banerjee, XML Schemas in Oracle XML DB, VLDB 2003 in Berlin, Sep 2003.

[10] http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html

[11] http://monetdb.cwi.nl/xml/index.html

[12] http://www.comp.nus.edu.sg/~ebh/XOO7.html

[13] http://www.eecs.umich.edu/db/mbench/

[14] Raugh Ramakrishnan et al., "Database Management Systems(3rd ed)," McGrawHill

[15] 김재욱, 이상원, 객체관계 DBMS를 이용한 XBench 벤치마크 성능 평가 및 분석, KDBC 2004 학술발표 논문집, 제20권, 제 2호, pages 83-90.

김 재 욱(Jae-Uk Kim) 준회원

1992년 2월 : 홍익대학교 졸업(학사)
2003년 3월-현재 성균관대학교 VLDB 연구실 석사과정

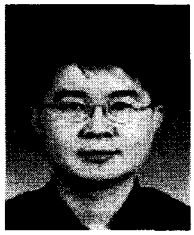
<관심분야> XML DBMS, 객체 관계형 DB

송 용 호(Yong-Ho Song) 준회원

1999년 2월 : 계명대학교 졸업(학사)
2004년 3월-현재 성균관대학교 VLDB 연구실 석사과정

<관심분야> XML DBMS, Query Optimization

이 상 원(Sang-Won Lee) 정회원



1991년 서울대학교 졸업 (학사)
1994년 서울대학교 졸업 (석사)
1999년 서울대학교 졸업 (박사)
1999년~2001년 한국 오라클
2001년~2002년 이화여대 연구교수
2002년~현재 : 성균관대학교 조교수

<관심분야> Database, Flash Memory