기업간통합 XML 메시지의 기록과 색인을 위한 저장 방식

(A Storage Scheme for Logging and Indexing B2Bi XML Messages)

송 하 주 † 김 창 수 † 권 오 흠 †
(Ha-Joo Song) (Chang-Su Kim) (Oh-Heum Kwon)

요 약 기업간통합(business-to-business integration; B2Bi) 시스템은 XML 메시지의 송수신 내역을 고속으로 기록하고 검색할 수 있는 메시지 저장시스템이 필요하다. XML 전용데이타베이스 시스템 또는 XML 데이타타입을 지원하는 관계형데이타베이스는 도입 비용과 기능의 과도함 때문에 메시지 저장시스템으로는 적합하지 않다. XML 데이타를 관계형데이타베이스의 레코드로 분산시켜 저장하는 저장 방식 또한 수행 성능이 떨어지거나 구현이 복잡하므로 사용하기 어렵다. 이에 본 논문은 관계형데이타베이스를 사용하여 메시지 저장시스템을 구현하기 위한 단순한 구조의 메시지 저장 방식을 제안한다. 제안하는 저장 방식은 메시지 타입별로 색인필드를 등록하고 색인필드를 통해서만 메시지를 검색한다. 따라서 XQL과 같은 강력한 검색 기능은 지원하지 못하나 비교적 단순한 저장 구조만으로도 고속의 메시지 기록과 제한된 형태의 메시지 검색이 가능하다. 제안하는 저장방식을 구현하기 위해서는 세 가지의 데이타베이스 스키마를 사용할 수 있으며 실험 평가를 통해 이들 세 가지의 스키마가 가지는 성능상의 장단점을 평가한다.

키워드 : XML, 기업간통합, 데이타베이스

Abstract A B2Bi system needs a message storage subsystem that efficiently logs and searches XML messages which have been sent from orreceived by it. XML database systems and XML enabled relational databases systems are not adequate as a message storage system because of their expensiveness and excessiveness in functionality. Storage schemes that split XML messages into database records are also unacceptable because of either low performance or implementation hardness. In this paper, we propose a storage scheme that can be applied to implement a message storage system based on a relational database system. In this scheme, messages are examined only through the index fields that have been registered for each message types. Therefore, the proposed storage scheme cannot support such a powerful search facility like XQL, but it provides high performance message logging and restricted search facility. There are three alternative database schemas to store the index fields. This paper compares the advantages and disadvantages of the three schemas through experimental tests.

Key words : XML, B2B integration, database

1. 서 론

1.1 XML 메시지의 저장과 검색

기업간협업(B2B collaboration)을 위한 소프트웨어 기술은 인터넷이 등장하기 이전부터 전자문서교환(EDI: electronic data interchange)와 같은 방식으로 사용되었다. 그러나 이러한 기술은 복잡하고 비용이 많이 들어 일부기업에서나 사용할 수 있었다. 인터넷의 등장으로 저렴한 비용으로 광대역의 통신이 가능해지고 XML 메시지(이하 메시지로 표기)를 사용함으로써 협업을 위한 응용프로그램을 과거보다 쉽게 개발할 수 있게 되었다 [1,2]. 이와 같은 기술은 기업간통합(B2Bi: business-to-business integration)으로불리며 웹서비스(web-services) 또는 ebXML과 같은 표준화된 기술이 점차 적용되어 가고 있다.

† 정 회 원 : 부경대학교 전자컴퓨터정보통신공학부 교수
hajusong@pknu.ac.kr
cskim@pknu.ac.kr
ohkwon@pknu.ac.kr
논문접수 : 2005년 1월 21일
심사완료 : 2005년 6월 27일

기업간통합시스템(B2Bi system)은 내부 시스템으로부터 추출된 데이타를 바탕으로 메시지를 작성하고 그것을 인터넷을 통해 거래 상대방의 기업간통합시스템으로 전달하거나(송신 메시지), 반대로 상대방으로부터 전송 받은 메시지(수신 메시지)에서 필요한 데이타를 추출하여 내부 시스템에 반영한다. 송수신 되는 메시지는 주문서, 송장, 출하요청 등과 같은 비즈니스 상의 중요한 문서에 해당되므로 추후 검증이 필요한 경우를 대비하여 메시지 원본을 저장한다. 저장된 메시지들은 또한 시스템의 동작상황을 감시하거나 성능을 높이기 위한 통계정보를 제공할 수도 있다(그림 1 참조). 따라서 기업간통합시스템은 내부적으로 메시지를 기록하고 추후 관리자에 의해 검색할 수 있는 메시지 저장시스템이 필요하다. 다음은 이러한 메시지 저장시스템에서 제공되어야 하는 기능적 특징에 대해 설명한다.

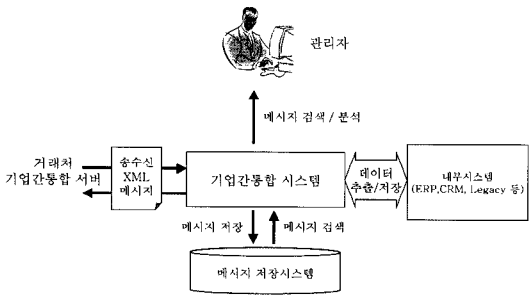


그림 1 XML 메시지의 저장과 검색

첫째, 메시지 저장장치는 단순한 XML 검색 기능만을 제공하면 된다. 기업간통합 XML 메시지의 경우에는 미리 정의된 종류의 메시지만을 사용하여 송수신을 한다. XQL과 같은 일반적인 질의문을 사용하여 임의의 XML 데이타에 대한 다양한 조건식을 사용하여 검색하는 일은 거의 없다. 또한 텍스트검색(text retrieval)에서처럼 특정 키워드를 이용하여 검색하는 일도 거의 없다. 다만 각각의 메시지 종류별로 특정 위치의 태그(tag) 또는 속성(attribute) 값을 사용하여 검색하는 것이 일반적이다.

그림 2는 기업간통합을 위해 사용되는 주문서(order) XML 메시지의 예를 보인 것이다. 기업간통합시스템에서는 주문서 내의 임의의 태그를 사용하여 검색을 수행하는 경우는 흔치 않으며 그림에서 타원으로 표시된 몇 개의 태그들을 사용하여 검색하는 것이 일반적이다.

둘째, 저장장치를 통해 저장되는 메시지 원문은 갱신되지 않는다. 메시지는 기존 거래방식에서의 문서와 같은 역할을 수행하는 것이므로 추후 갱신이 될필요가 없다. 셋째, 메시지를 고속으로 기록할 수 있어야 한다. 상거래를 나타내는 메시지의 송수신이 일어날 때 기록되

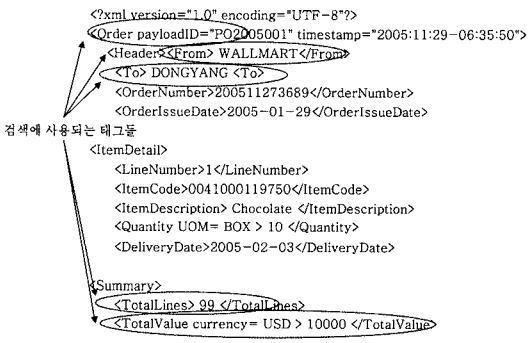


그림 2 XML 메시지의 예

는 것이므로 트랜잭션이 매우 빈번하게 일어나는 상황에서도 잘 동작할 수 있어야한다. 넷째, 메시지 저장을 위해 사용될 데이타베이스는 내부시스템에서 사용하는 데이타베이스 시스템을 그대로 사용하는 경우가 많다. 따라서 현재 널리 사용되고 있는 관계형데이타베이스 시스템(relational database management system)를 사용하게 될 가능성이 높다.

이와 같은 특징들은 XML 전용데이타베이스 또는 관계형데이타베이스 시스템에서 제공하는 XML 데이타 타입을 이용하여 메시지 저장시스템을 구현할 필요가 없거나 불가능함을 의미한다. 이러한 시스템들은 셋째 특징과 네째 특징으로 인해 사용할 수가 없는 경우가 많다. 또 첫째 또는 둘째 특징으로 인해 굳이 사용할 필요가 없다. 따라서 기업간통합시스템은 관계형데이타베이스를 사용하여 메시지 저장시스템을 구현하는 것이 일반적이라 할수 있다.

이에 본 논문에서는 관계형데이타베이스를 사용하여 기업간통합 메시지를 저장하고 검색할 수 있는 단순한 구조의 메시지 저장 구조를 제안한다. 제안하는 저장구조는 검색에 사용될 색인 필드를메시지 종류별로 미리 등록하고 송수신 되는 메시지의 내용 중 색인필드만을 데이타베이스에 저장한 후 그것들을 바탕으로 제한적인 메시지 검색 기능을 제공한다. 색인필드를 저장하기 위해서는 세가지의 데이타베이스 스키마를 사용할 수 있다. 이들은 데이타베이스 시스템이 지원하는 스키마 변경 기능의 지원 수준과 검색 기능의 사용빈도, 저장공간의 사용량에 따라 기능의 지원 여부 또는 성능이 달라지게 되는데 실험을 통해 각 스키마의 장단점을 비교한다.

논문은 다음과 같이 구성된다. 이어지는 2절에서는 XML 메시지의 저장과 관련한 기존 연구에 대해 간략히 살펴본다. 3장에서는 제안하는 저장기법과 그것을 구현하기 위한 데이타베이스 스키마들에 대해 설명한다. 4장에서는 제안된 세 가지의 스키마에 대해 합성데이타를 이용한 성능 비교 결과를 제시한다. 그리고 5장에서

결론을 맺는다.

2. 관련연구

XML 데이타를 저장하고 검색하기 위해최근 수년 동안 활발한 연구가 진행되고 있다[4-10]. 연구방향은 크게 (1) 텍스트(text) 파일 저장방식 (2) 관계 DTD 방식 (3) 간선(edge) 방식 (4)속성(attribute) 방식 (5) 객체(object) 방식으로 구분해 볼 수 있다[9,10].

(1) 텍스트 파일 저장 방식은 텍스트 파일에 XML 데이타를 저장하고 메인 메모리상에서 XML 파일을 파싱(parsing)하여 검색을 지원한다. 그러나 파싱에 의존한 검색만으로는 성능이 떨어지게 되므로 (parent_offset, tag) → child_offset, child_offset → parent_offset, (tag/attribute name, value) → tag_offset 등의 인덱스를 사용하여 성능을 높일 수 있다. 여기서 parent_offset, child_offset, tag_offset은 텍스트 문서 내에서 부모태그와 자식태그, 해당 태그의 문서 내 오프셋(offset)을 의미하는 것이다. 이 방식은 XML 데이타에 갱신이 자주 일어나지 않는 경우에 우수한 성능을 보인다.

(2) 관계 DTD 방식은 [10]에서 공유-인라인(shared-inline) 방식으로 제안된 방식이다. 이 방식은 이름이 같은 XML 태그들을 관계형데이타베이스의 테이블에 대응시키고 태그간의 부모-자식 관계를 키와 외래키 관계로 나타내는 방법이다. 자식이 없는 태그 또는 속성들은 모두 부모 태그에 해당하는 테이블의 컬럼으로 변환된다. 그림 3은 그림 2의 XML 메시지를 (2)방식을 사용하여 저장한 모습을 나타낸 것이다.

(3) 간선방식은 위의 (2)방식에서 여러 개의 테이블로 나눠 저장하던 것을 하나의 테이블에 모두 저장하는 것으로 볼 수 있다. 그림 4는 그림 2의 메시지를 간선방식을 사용하여 저장한 예를 보인 것이다. 각각의 레코드는 태그 또는 속성을 나타내는 것으로 SourceID가 부모 태그(또는 속성)를 나타내며 TargetID는 연결된 자식 태그(또는 속성)을 나타낸다. tag는 해당되는 레코드의 태그(또는 속성) 이름을 나타낸 것이다. ordinal은 동일한 부모에 연결된 자식들간의 순서를 나타내는데 ordinal이 0인 것은 속성에 해당된다. TargetID가 0인 것은 텍스트를 갖는 태그 또는 속성에 해당한다.

Order 테이블

ParentID	ID	payloadID	timestamp
1	2	PO2005001	2005:11:29-06:35:50

Header 테이블

ParentID	ID	From		OrderIssueDate
2	3	WALLMART		2005-01-29

ItemDetail 테이블

ParentID	ID	LineNumber		DeliveryDate
2	4	1		2005-02-03
2				

⋮

Summary 테이블

ParentID	ID	TotalLines
2	1000	99

TotalValues 테이블

ParentID	ID	currency	TEXT
1000	1001	USD	10000

그림 3 공유-인라인 방식을 사용한 XML 메시지의 저장

간선 테이블

SourceID	tag	ordinal	targetID	TEXT
1	Order	1	2	NULL
2	payloadID	0	0	PO2005001
3	timestamp	0	0	2005:11:29-06:35:50
2	Header	1	4	NULL
4	From	1	0	
4	To	2	0	
4	OrderNumber	3	0	
4	OrderIssueDate	4	0	
2	ItemDetail	2	101	NULL
101	LineNumber	1	0	1
101	ItemCode	2	0	0041000119750
2	Summary	3	201	NULL

그림 4 간선방식에 의한 XML 메시지의 저장

(4) 속성(attribute) 방식은 간선방식에서 tag의 이름이 같은 레코드들을 분리해 내서 별도의 테이블로 만들어내는 방식이다. 따라서 테이블 이름으로부터 tag를 유추할수 있으므로 tag를 레코드에 기록할 필요는 없는 장점이 있다. 반면 XML 메시지의 태그(또는 속성)의 이름마다 하나의 테이블을 사용해야 하므로 테이블의 수가 지나치게 많아질 수 있다.

(5) 객체 방식은 XML 메시지를 객체에 대응시키고 메시지내의 태그들은 경량급 객체(light-weight object)에 대응시킨다. 속성값은 경량급 객체의 데이타 필드로 추가되고 텍스트 데이타는 경량급 객체 내에 저장되거나 별도의 경량급 객체로 분리 저장된다.

[9]의 실험결과에 따르면 관계형데이타베이스에 DTD를 같이 사용한 (2)방식이 저장공간과 성능측면에서 범용적으로는 가장 좋은 성능을 나타내며 DTD내에 사이클(cycle)을 사용한질의검색에는 객체 방식을 사용하는 방식이 우수하다. 그리고 XML 데이타에 대한 갱신이 드문 경우에는 텍스트 파일을 이용하여 XML 데이타를 저장하고 데이타베이스를 인덱스 구조로 사용하는 방식도 객체 방식과 사용하는 것과 유사한 성능을 제공한다.

이러한 XML 저장시스템에 대한 연구들은 모두

XML 데이타를 저장시스템에 기록하고 XQL과 같은 일반적인 질의어를 사용하여 검색하는 경우에 그 실행성능을 높이기 위한 연구들이다. 따라서 일반적인 XML 문서에 대한 저장시스템으로는 적합하나 메시지 저장장치로 사용하기에는 불필요한 기능이 많다. 그리고 XML 저장시스템은 데이타 삽입이 매우 빈번하게 일어나는 환경에서 요구되는 충분한 삽입성능을 제공하지 못할 수 있다. (2)~(4) 방식은 조인을 사용해야 하기 때문에 메시지 원문을 재구성하는 오버헤드가 매우 크다.

따라서 본 연구에서는 기존의 (1) 텍스트 파일 저장 방식을 변형한 저장 방식을 사용했다. 이 방식을 택한 이유는 메시지 원문에 대한 조회와 메시지 자체에 대한 기록이 가장 빨리 이루어 질 수 있기 때문이다[9]. 한편, (1) 방식에서는 메시지 내의 모든 태그들을 사용한 검색이 가능하나 제안하는 방식은 관리자에 의해 미리 등록된 일부 태그 들에 대한 검색만을 지원한다. 따라서 검색 기능은 제약을 받으나 메시지 기록시 색인 정보를 삽입하는 과정을 최소화 할 수 있고 그에 따라 기록 성능을 더욱 높일 수가 있다. 제안하는 방식은 또한 색인에 필요한 정보를 XPath를 이용하여 관리자로부터 입력 받는다. 따라서 저장시스템에서 메시지내의 태그들의 오프셋(offset)을 계산하는 것과 같은 복잡한 기능을 수행하지 않으므로 (1)에서와 같은 다수의 인덱스를 사용하지 않는다. 또한 기존의 (2)~(5)방식과 같은 복잡한 스키마 처리를 동반하지 않으므로 단순한 구조의 메시지 저장시스템을 구현할 수 있다.

3. 기업간통합 XML 메시지를 위한 저장방식

3.1 XML 메시지에 대한 색인 정보

먼저 저장되는 메시지에서 추후 검색을 위해 사용되는 태그(또는 속성)을 '색인필드(index field)'로 정의한다. 그 위치는 메시지 타입별로 XPath를 사용하여 미리 등록된다. 제안하는 저장방식은 메시지 원문은 파일시스템에 저장하고 색인필드 값을 비롯한 메시지 관련 정보는 관계형데이타베이스에 저장한다. 메시지 원문에 대해서는 갱신이 일어나지 않고 원문의 임의의 위치에 대해 XPath를 사용하여 검색하는 기능은 필요 없기때문에 파일 시스템에 저장하는 것이 유리하다. 또 파일 시스템을 사용하면 XML관련 응용프로그램을 연동하기에도 편리하다.

그림 5는 데이타베이스에 저장되어야 할 데이타의 구조를 확장개체관계모델(extended entity-relationship model)로 나타낸것이다. 그림에서 MSG_TYPE은 기업간통합 시스템에서 사용되는 메시지 타입과 관련된 메타정보를 저장하는 개체이다. 메시지 타입별로 등록되는 정보는 다음과 같다.

- name: 메시지 타입의 이름
- 메시지 타입 식별을 위한 조건
  - rootTag: 루트 태그의 이름
  - url: XML document type URL
  - constraint1, constraint2, constraint3: XPath를 사용한 조건식
- callback: 해당 메시지의 수신시 적용될 처리루틴
- 색인필드별 정보
  - name: 색인필드의 이름
  - path: XPath로 표현된XML 메시지 내에서 해당 색인필드의 위치
  - datatype: 색인필드의 데이타 타입

하나의 메시지 타입에 대해 여러 개의색인필드가 존재할 수 있으므로 색인필드에 관련된 정보는 INDEX_FIELD라는 별개의 개체를 사용한다. XML_MESSAGE 개체는 모든 메시지들에 대해 공통적으로 사용되는 공통 메시지 필드를 유지한다. 여기에는 다음과 같은 정보가 포함된다.

- msgId: 메시지 식별자 (모든 메시지에 대해 유일, 송수신시 생성)
- umsgId: 메시지 내에서 추출된 메시지 식별자(동일 거래처에 대해서는 유일)
- archi_time: 송수신 시각
- sender/receiver: 메시지 송/수신자
- loginId: 메시지 송수신시 로그인한 사용자의 아이디
- filepath: 파일시스템상에 저장된 메시지 원문의 위치 (file path)
- bound: 송신일 경우 true, 수신일 경우 false

이중에서 umsgid, sender, receiver 등은 저장되는 XML 메시지로부터 XPath를 사용하여 추출되는 색인필드이고 나머지는 메시지 송수신 환경으로부터 전달받는 값들이다. XML_MESSAGE 개체의 하위 개체로표시된 ORDER, INVOICE, INV_REPORT 등이 실제 메시지에서 추출된 인덱스 필드의 값을 저장하기 위한 개체이다. 따라서 이들 개체들은 고정된 것이 아니며 기업간통합 시스템에서 사용하는 메시지 타입이 변경되면 생성/삭제 될 수 있다. XML_MESSAGE 개체와 이들 개체들간에 상속관계를 사용한 것은 다음과 같이 두 가지 방식의 검색이 모두 가능해야 하기 때문이다.

- 메시지 타입에 상관없이 msgid, timestamp, sender, receiver 등에 대한 조건만으로 메시지들을 검색하는 경우
- 메시지 타입과 색인필드를 함께 사용하여 메시지를 검색하는 경우

그림 5의 저장구조 모델링에서 MSG_TYPE과 INDEX_FIELD 개체집합은 관계형데이타베이스의 테이블로 변환하고 참조키(foreign key)를 사용하여 상호 참조관

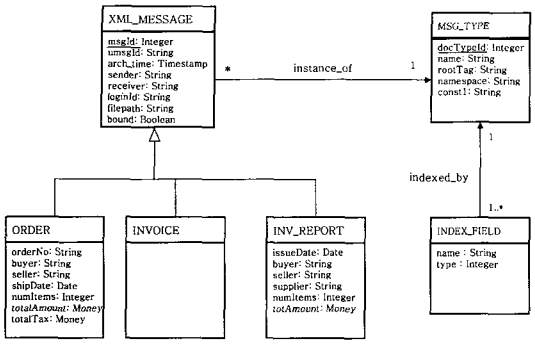


그림 5 확장 개체-관계 모델로 표현된 메시지 저장 구조

계를 표현할 수가 있다. 그러나 XML_MESSAGE 개체와 Order, Invoice 등의 메시지 타입별 상속관계는 관계형데이타베이스를 이용해서 나타낼 수 없다. 이에 본 논문에서는 관계형 데이타베이스를 이용하여 상속을 표현하기 위한 저장스키마들를 제시하고 기업간통합 시스템에 사용되는 관계데이타베이스 시스템의 특성에 따라 선택해서 사용할 것을 제안한다. 이어지는 장에서는 세 가지 스키마를 차례로 설명한다.

3.2 저장스키마1(storage scheme1): 메시지 타입별로 테이블을 사용하는 스키마

그림 6은 저장스키마1을 보인 것이다. 이 방식은 메시지 타입별로 데이타베이스의 테이블을 하나씩 대응시킨다. MSG_TYPE 테이블과 INDEX_FIELD 테이블은 그림 5의 MSG_TYPE 개체집합과 INDEX_FIELD 개체집합에 각각 해당한다. INDEX_FIELD의 레코드는 외래키(foreign key)인 msgTypeId 컬럼을 통해 MSG_TYPE 레코드와 연결된다.

메시지 타입이 새로 추가될 때마다 MSG_TYPE 테이블에 해당 메시지 타입에 대한 정보를 포함하는 새로운 레코드가 추가된다. 또한 메시지 타입의 이름에 해당하는 테이블이 생성되고 해당 타입의 색인필드에 대응되는 컬럼들이 테이블내에 생성된다.

XML_MSG 테이블은 그림 5의 XML_MESSAGE에 대응하는 테이블로 저장되는 모든 메시지에 대해 공통적으로 추출되는 정보를 기록한다. 따라서 하나의 메시지를 기록하기 위해서는 XML_MSG 테이블과 해당되는 타입의 테이블에 각각 레코드가 삽입된다. 타입에 상관없이 저장된 메시지를 검색하는 경우에는 XML_MSG 테이블만 검색한다. 메시지 타입을 포함한 검색시에는 해당되는 타입의 테이블만을 사용하거나 msgId 필드를 이용하여 두 테이블을 조인하여 검색한다. 다음은 주어진 검색 조건에 해당하는 메시지를 찾기 위한 SQL문을 나타낸 것이다.

- 검색 조건:
송신된 주문서(Order) 메시지들 중에서 주문 총액이 1000$ 이상인 것의 수신자와 메시지 송신시각을 찾아라

- 대응되는 SQL문:

```
SELECT receiver, arch_time
FROM XML_MSG JOIN  ORDER ON msgId
WHERE bound = TRUE
AND totAmount >= 1000
```

예 1 저장스키마1에서 메시지 검색의 예

이 방식은 메시지 타입의 추가 또는삭제가 일어나는 경우 대응되는 테이블이 새로 생성되거나 삭제된다. 마찬가지로 메시지 타입내의 색인필드가 추가(삭제 또는 재명명(rename))되는 경우 해당되는 테이블의 컬럼이 추가(삭제 또는 재명명)되어야 한다. 따라서 이 방식은 스키마 진화(schema evolution)가 잘 지원되는 데이타베이스에서는 문제가 없지만 그렇지 않은 경우에는 색인필드의 추가(삭제 또는 재명명)이불가능할 수도 있는 방식이다.

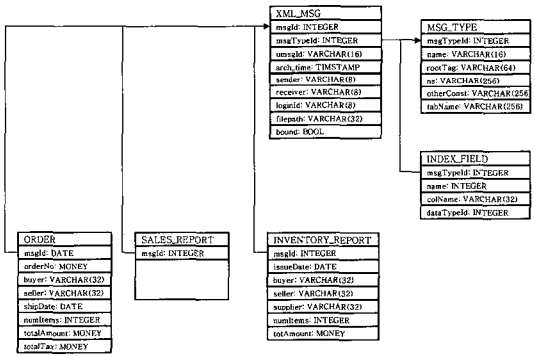


그림 6 저장구조1: 메시지 타입별로 테이블을 사용

3.3 저장스키마2(storage scheme2): 단일 테이블에 모든 메시지 정보를 저장하는 스키마

두번째 방식은 타입에 상관없이 모든 색인필드를 하나의 테이블(XML_MSG)에 저장하는 방식이다. 따라서 XML_MSG 테이블은 모든 메시지 타입에서 사용되는 색인필드에 대응하는 컬럼을 유지해야 한다. 반면 실제 메시지가 기록될 때는 해당 메시지 타입에 속한 색인필드에 대응하는 컬럼들만이 사용될 것이다. 그러므로 공통 색인필드에 해당하는 컬럼을 제외한 모든 컬럼들은 Nullable로 정의된다. 이것은 XML_MSG 테이블의 레코드들에 대해 상당수의 컬럼이 널값만을 가지게 될 것임을 의미한다. 이 방식은 메시지 하나에 대해 정확히 하나의 레코드만이 사용된다. 예1에서 사용된 검색을 위

해서는 예2와 같은 SQL문을 사용할 수 있다.

```
• 예1의 검색조건에 대응되는 SQL문:
    SELECT receiver, arch_time
    FROM XML_MSG JOIN MSGTYPE ON msgTypeId
    WHERE bound = TRUE
    AND MSG_TYPE.name = 'ORDER'
    AND totAmount >= 1000
```

예 2 저장스키마2에서 메시지 검색의 예

수행성능을 높이기 위해 MSG_TYPE 정보와 INDEX_FIELD 정보는 기업간통합시스템에 캐싱(caching)하는 보통이다. 따라서 만일 'ORDER' 메시지 타입의 msgTypeId가 1이라 할 때 위의 SQL문은 다음의 예 3과 같이 단순화할 수 있다.

```
• 예1의 검색조건에 대응되는 SQL문:
    SELECT receiver, arch_time
    FROM XML_MSG
    WHERE bound = TRUE
    AND XML_MSG.msgTypeId = 1
    AND totAmount >= 1000
```

예 3 저장스키마2에서 msgTypeId를 캐싱한 경우 메시지 검색의 예

두번째 저장 방식 또한 첫번째 방식과 마찬가지로 메시지 타입이나 색인필드의 추가, 삭제, 재명명을 위해서는 데이타베이스의 스키마를 변경해야 한다.. 따라서 기업간통합 시스템이 사용하는 관계형데이타베이스가 해당되는 기능을 제공하는 경우에만 사용할 수가 있다.

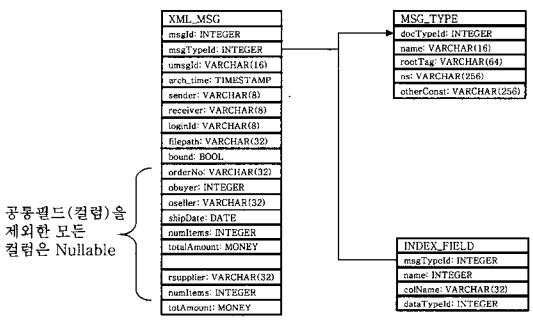


그림 7 저장구조2: 단일 테이블 저장 방식

3.4 저장스키마3(storage scheme3): 메시지 테이블과 색인필드 테이블을 사용하는 스키마

세번째 방식은 메시지별 색인 필드의 값을 하나의 테이블에 저장하는 방식이다. XML 메시지에 대해 공통필드는 XML_MSG 테이블에 기록되고 메시지 타입별 색인필드는 INDEX_FIELD_VAL 테이블에 저장된다. 단, 각각의 색인필드 값별로 하나의 레코드가 사용된다. 만약 Order 메시지 타입에 7개의 색인필드가 있었다면 최대 7개의 레코드가 INDEX_FIELD_VAL 테이블에 저장된다. 제안된 세가지 방식 중에서 메시지 하나당 사용되는 레코드의 개수 측면에서는 가장 많은 레코드를 사용하게 될 것이다.

INDEX_FIELD_VAL 테이블의 msgId 컬럼을 통해 관련된 XML_MSG 레코드를 찾을 수 있고 filedId 컬럼을 통해 대응하는 색인필드에 대한 정보를 검색할 수 있다. 메시지 타입 및 색인필드 정보를 캐쉬한 것으로 가정하고 ORDER 메시지 타입의 msgTypeId =1, totAmount 색인필드의 fieldTypeId = 8일때, 예1의 검색을 위해서는 다음과 같은 SQL문을 사용할 수 있다.

```
• 예1의 검색조건에 대응되는 SQL문:
    SELECT sender, arch_time
    FROM XML_MSG JOIN INDEX_FIELD_VAL ON msgId
    WHERE bound = TRUE
    AND INDEX_FIELD_VAL.msgTypeId = 1
    AND INDEX_FIELD_VAL.fieldTypeId = 8
    AND INDEX_FIELD_VAL.valMoney >= 1000
```

예 4 저장스키마3에서 메시지 검색의 예

INDEX_FIELD_VAL 테이블의 각 컬럼은 데이타베이스에서 지원하는 기본 데이타타입(primitive data type)에 해당한다. 여러 개의 컬럼이 정의되었더라도 실제 사용될 때는 색인필드의 타입에 대응하는 컬럼만 값이 할당되고 나머지 컬럼은 사용되지 않을 것이므로 모든 컬럼은 Nullable로 정의되어야 한다. INDEX_FIELD_VAL의 레코들은 사용되는 컬럼을 제외한 모든 컬럼이 널값을 갖게 된다.

이 방식은 가장 단순한 구조의 저장 방법이다. 메시지 타입이 추가 또는 삭제되더라도 데이타베이스의 스키마

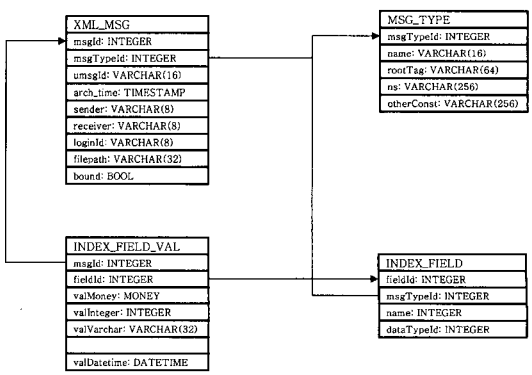


그림 8 저장구조3: 색인필드값 테이블을 사용하는 방식

에는 아무런 영향을 미치지 않는다. 따라서 데이타베이스에서스키마 변경 기능을 지원하지 않더라도 사용할 수 있는 방법이다. 반면 INDEX_FIELD_VAL의 레코드는 하나의 하나의 컬럼만이 사용되고 나머지는 모두 NULL 값을 갖게 되므로 저장공간이 낭비된다. 또한 valVarChar의 경우에는 미리 정의된 크기 이상의 크기를 갖는 색인필드를 사용하기 어렵다.

4. 저장 스키마별 성능 비교

이 장에서는 제안된 세가지 저장방식들을 실험을 통해 성능과 특성을 비교한다. 성능 비교는 백만(10^6) 개의 메시지를 사용했을 때, 저장공간의 사용량, 삽입성능, 검색성능의 세가지 측면에서 수행하였다. 메시지의 공통필드와 색인필드는 모두 합성된 데이타 값을 사용하였다. 저장공간의 사용량은 각 저장방식에 따라 저장을 하였을 때, 실제 레코드들이 차지하고 있는 공간의 크기와 인덱스가 차지하는 공간을 측정하였다. 삽입성능의 경우에는 백만개의 메시지를 순차적으로 모두 기록하는데 소요된 시간을 측정하였다. 검색성능은 콜드버퍼(cold buffer) 상태에서 다음과 같은 세가지 질의를 처리하는데 소요된 시간을 각각 측정하였다.

- 질의1(Q1): 전체 메시지의 1%(10^4)에 해당하는 메시지들에 대해 주어진 메시지식별자(msgId)에 대응하는 메시지를 검색하고 공통필드 값을 출력
- 질의2(Q2): 주어진 sender와 umsgId를 갖는 메시지를 검색하되 공통필드의 값을 출력
- 질의3(Q3): 지정된 메시지 타입에 속하고 해당 타입의 지정된 색인필드 값을 갖는 메시지를 검색하되 공통필드만을 출력

4.1 실험 설정

본 실험을 위해 클라이언트와 서버 컴퓨터를 각각 1대씩 사용하였으며 이들은 사양은 다음과 같다.

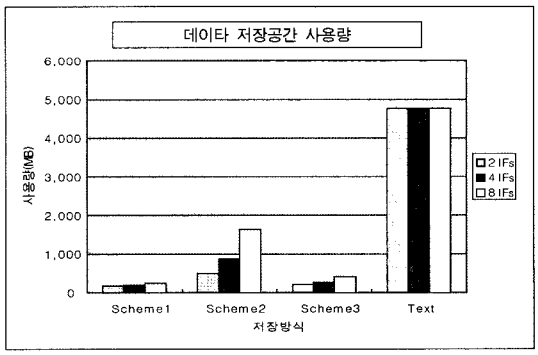
- 하드웨어(서버) - CPU: Intel Pentium 4 2.6GHz, RAM: 1GB, HDD: 80GB 15,000 RPM
- 하드웨어(클라이언트) - CPU: Intel Pentium 4 2.6GHz, RAM: 1GB
- 운영체제: Microsoft Windows 2000 Server
- 데이타베이스 시스템: Microsoft SQL Server 2000, 1GB 메모리중 512MB만 사용하도록 설정

실제 기업간통합시스템이 사용되는 환경은 다양한 메시지 타입과 색인필드가 사용될 수 있으므로메시지 타입의 개수, 메시지당 색인필드의 개수를 중심으로 성능을 평가하였다. 메시지 타입의 종류는 4종, 16종, 64종으로 구분하여 실험한다. 메시지의 개수는 백만개로 고정되어 있으므로 타입이 늘어나면 하나의 타입에 속하는 메시지의 개수는 줄어들게 된다. 공통 메시지 필드는 msgId 컬럼을 포함하여 8개로 하였다. 메시지당 색인필드의 수는 2개, 4개, 8개로 실험하였는데 색인필드의 개수가 늘면 메시지 자체의 크기는 증가하게 된다. 각 필드는 정수형(INTEGER) 또는 문자열형(VARCHAR(16)) 만을 사용했다. 따라서 Scheme3에서 INDEX_FIELD_VAL 테이블의 컬럼은 valInt와, valVarChar만 사용하였다.

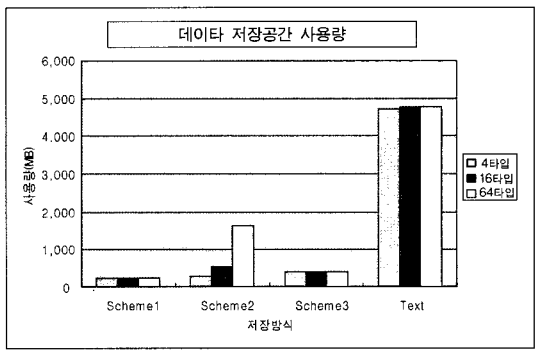
4.2 실험 결과

그림 9는 모든 메시지를 저장하는데 소요된 저장공간 사용량을 비교한 것이다. Scheme1, Scheme2, Scheme3는 각각 저장방식1, 저장방식2, 저장방식3을 의미하고 Text는 2장에서 설명된 관련연구 중에서 제안하는 저장방식과 가장 유사한 형태인 텍스트 저장방식에 대한 실험결과를 나타낸 것이다. 그림 9 (a)는 타입당색인필드가 2개, 4개, 8개로 늘어남에 따라 저장공간의 사용량을 보인 것이고 그림 9 (b)는 8개의 색인필드를 사용하되 메시지 타입의 수가 4개, 16개, 64개로 증가함에 따른 저장공간의 사용량을 보인 것이다.

그림 9 (a)에서 보여주듯이 타입당 색인필드의 개수가 증가할수록 모든 메시지 하나당 저장되는 데이타의 양이 증가하므로 Text를 제외한 모든 방식에서 저장공



(a) 타입당 색인필드 개수에 따른 저장공간 사용량

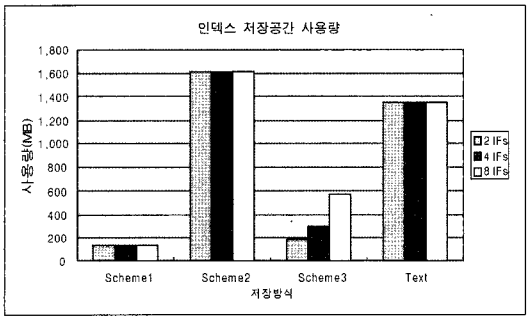


(b) 메시지 타입의 개수에 따른 저장공간 사용량

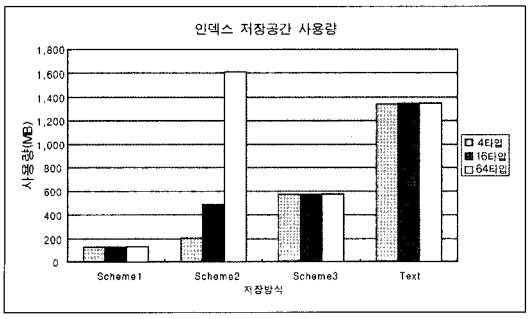
그림 9 데이타 레코드 저장공간 사용량

간의 사용량이 늘어나게 된다. Scheme1과 Scheme2의 경우에는 레코드의 크기가 늘어나는 것이고 Scheme3의 경우에는 INDEX_FIELD_VALUE 테이블의 레코드 개수가 늘어나는 것으로 볼 수 있다. 특히 Scheme2의 경우에는 실제 메시지 타입과 관련 없는 색인필드 또한 NULL 값으로 모두 저장되어야 하기 때문에 레코드의 크기가 다른 방식에 비해 급속히 늘어나게 된다. 동일한 이유로 Scheme2는 그림 9 (b)에서 보듯이 메시지 타입이 증가하면 다른 방식에 비해 훨씬 많은 저장공간을 소모하게 된다. Scheme1과 Scheme3는 메시지 타입이 증가하더라도 저장공간의 사용량에는 변화가 거의 없는데 그것은 두 방식 모두 저장되는 레코드의 크기에는 변화가 없고 저장되는 테이블만이 바뀌거나(Scheme1) msgTypeId만 변경되기(Scheme3) 때문이다. 어느 경우에서나 Scheme1이 가장 적은저장공간을 사용하는데 그것은 색인필드를 여러 개의 레코드로 나누어 저장하는 Scheme3보다 적은개수의 레코드만을 사용하는 때문인 것으로 볼 수 있다. Text는 메시지 내의 모든 태그에 대한 데이타를 기록하므로 메시지 타입의 수나 색인 필드의 개수에 영향을 거의 받지 않는 반면 저장공간은 가장 많이 사용한다.

그림 10은 각각의 메시지 타입별로 2개의 색인필드에



(a) 타입당 색인필드 개수에 따른 인덱스 저장공간 사용량

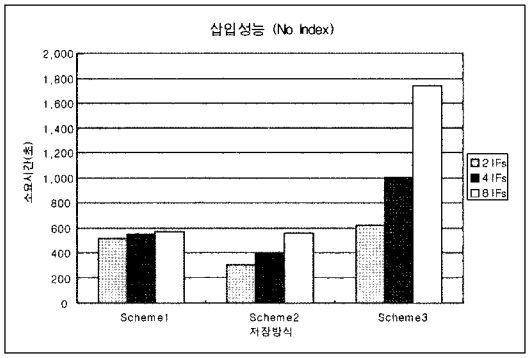


(b) 메시지 타입의 개수에 따른 인덱스 저장공간 사용량

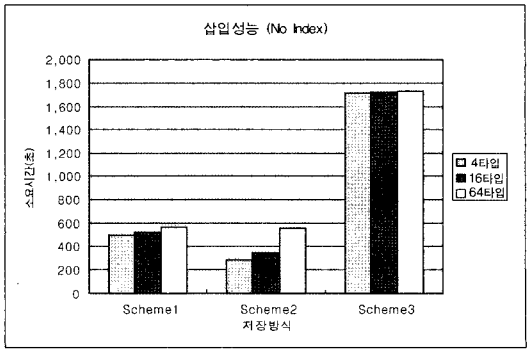
그림 10 인덱스 저장공간 사용량

대해 인덱스를 생성하였을 때, 인덱스 저장공간의 크기를 비교한 것이다. 어느 경우나 대체적으로 Scheme2가 가장 많은 인덱스 저장공간을 소비한다. Scheme2를 제외하고는 인덱스 저장공간의 크기 또한 대체적으로 데이타 레코드 저장공간과 일치하는 경향을 보인다. 이것은 색인필드의 개수를 늘리더라도 추가된 색인필드에 대해서는 인덱스를 사용하지 않으므로 Scheme2의 경우에는 인덱스 저장공간이 늘지 않기 때문인데 Scheme1도 동일한 이유로 색인필드의 개수에는 영향을 받지 않았다. 반면 메시지 타입의 개수가 늘어나는 경우, Scheme1은 전체 레코드가 여러 개의 테이블에 분산되어 저장되므로 인덱스 객체의 수는 늘지만 개개의 인덱스 크기는 줄게 되어 전반적인 저장공간의 사용량은 큰 변화가 없다. 그러나 Scheme2의 경우에는 인덱스 객체의 수는 늘되 각각의 인덱스에 저장되는 레코드의 수는 변함이 없으므로 인덱스 저장공간이 메시지 타입의 수에 비례하여 늘어나게 된다. Text는 모든 필드를 색인하는 것으로 볼 수 있으므로 색인 필드의 개수에 영향을 거의 받지 않는다.

그림 11은 모든 색인필드에 대해 인덱스를 전혀 사용



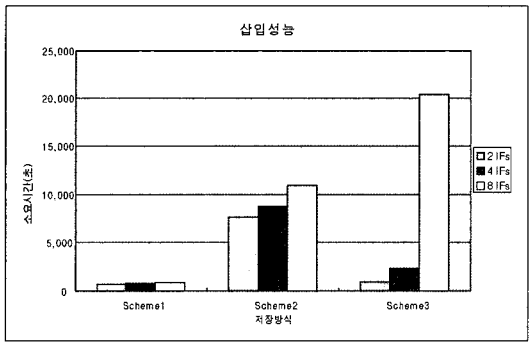
(a) 색인필드의 개수에 따른 삽입성능 비교



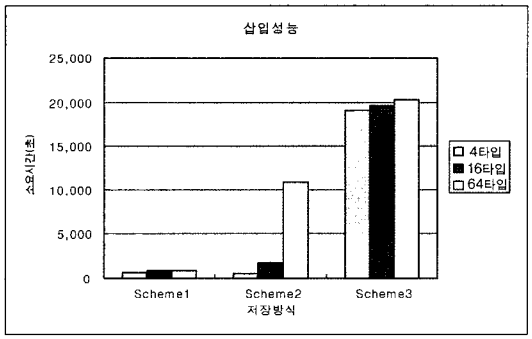
(b) 메시지 타입의 개수에 따른 성능비교

그림 11 레코드의 삽입성능 비교(인덱스를 사용하지 않은 경우)

하지 않고 모든 메시지를 삽입하였을 때 각 방식에 따른 삽입성능을 비교한 것이다.[1] 인덱스를 전혀 사용하지 않은 경우에는 모든 경우에 대해 Scheme2가 가장 우수한 삽입 성능을 나타내었다. 이것은 Scheme2에서는 타입별 색인필드의 개수나 타입의 개수에 상관없이 항상 하나의 레코드만을 삽입하기 때문으로 볼 수 있다. 메시지 타입당 색인필드의 개수가 증가하거나 메시지 타입의 수가 늘어나면 Scheme2는 레코드의 크기가 늘어나므로 대체적으로 삽입에 소요되는 시간이 증가하게 된다. 반면 Scheme3의 경우에는 타입별 색인필드의 개수가 증가하면 메시지 하나당 삽입해야 하는 색인필드값 레코드의 개수도 증가하기 때문에 삽입 성능이 급격하게 떨어지게 됨을 알 수 있다(그림 11 (a)). 이러한 성능저하는 색인필드값 레코드를 삽입할 때 마다 외래키에 대응하는 XML_MSG 테이블의 레코드가 존재하는지를 데이타베이스 시스템 내부적으로 매번 확인해야 하기 때문에 더욱 악화되는 것으로 파악할 수 있다. 반면 메시지 타입이 늘어나더라도 Scheme3는 삽입 성능에 큰 변화가 없는데(그림 11 (b) 참조) 그것은 타입이 늘어나도 삽입되는 레코드의 msgTypeId 필드 값만이 달라질 뿐이며 레코드의 크기에는 변화가 없기 때문이다. Text는 다른 방식에 비해 10배 이상의 시간을 소모했다.



(a) 색인필드의 개수에 따른 삽입성능 비교



(b) 메시지 타입의 개수에 따른 삽입성능 비교

그림 12 레코드의 삽입성능 비교(인덱스를 사용한 경우)

1) primary key로 정의되는 msgId와 같은 컬럼들에 대해서는 데이타베이스 시스템 수준에서 여전히 인덱스를 사용한다.

그림 12는 색인필드에 인덱스를 사용한 경우에 대해 삽입 성능을 비교한 것이다. Text는 이번에도 다른 방식보다 10배 이상의 시간을 소모한 관계로 그래프에 나타내지 않았다. 인덱스는 각 메시지 타입별로 2개의 색인필드에 대해서만 사용하였다. Scheme1은 타입당 색인필드가 늘거나 타입의 수가 늘어도 삽입성능에는 큰 영향을 받지 않고 다른 방식에 비해 성능 또한 가장 우수하다. 이것은 색인필드의 개수가 늘더라도 그것이 인덱스를 사용하는 색인필드가 아니면 크게 영향을 주지 않기 때문이다. 메시지 타입이 늘어나는 경우에도 사용하는 인덱스의 개수가 늘어날 뿐이며 레코드 하나를 삽입할 때 동시에 변경해야 하는 인덱스의 개수는 늘지 않기 때문인 것으로 판단된다. Scheme2의 경우에는 인덱스와 관련 없는 색인필드의 개수가 늘더라도 영향을 거의 받지 않으나 타입이 늘어나면 사용하는 인덱스의 개수가 늘면서 인덱스 필드들이 늘어나기 때문에 삽입 성능이 떨어지게 된다. Scheme3은 색인필드 테이블의 컬럼 개수(데이타 타입의 수)만큼 인덱스를 사용하게 된다. 즉, 데이타 타입별로 인덱스가 하나씩 존재하므로 색인필드의 개수를 늘린 만큼 인덱스의 갱신이 필요하다 따라서 필드의 개수가 많아질수록 삽입성능이 떨어지게 된다(그림 12 (a)). 반면 타입의 개수가 늘더라도 필드의 개수에 변화가 없으면 인덱스 갱신 회수가 늘지 않으므로 타입의 개수에는 크게 영향을 받지 않게 된다 (그림 12 (b) 참조).

그림 13은 64개 타입을 사용하여 모든 메시지를 저장한 다음 Q1질의를 수행한 결과인데 Scheme1, Scheme3, Scheme2 순의 검색 성능을 보인다. msgId를 이용한 검색은 XML_MSG 테이블의 주키(primary key)로 정의된 msgId에 대해 클러스터 인덱스를 사용하게 된다. 따라서 XML_MSG 테이블의 레코드 크기가 작을수록

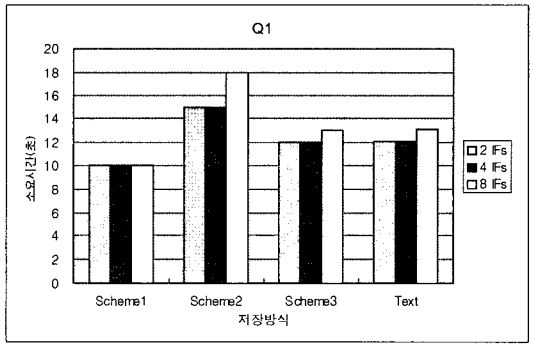


그림 13 메시지식별자(msgId)에 대한 검색 성능 비교

인덱스의 단말 페이지에 많은 수의 레코드가 포함되고 인덱스 트리의 높이가 낮아져서 우수한 검색성능을 보이게 된다. 동일한 이유로 메시지 타입 개수의 변화에 대한 검색성능 또한 그림 13과 유사한 결과를 나타내므로 결과 그래프는 생략하였다.

그림 14는 저장된 메시지들에 대해 Q2질의를 수행한 성능을비교한 결과이다. Q2 질의는 그 특성상 XML_MSG 테이블을 스캔(scan)하게 되므로 콜드 버퍼(cold buffer)의 경우에는 XML_MSG 테이블의 크기에 좌우된다.

Scheme1과 Scheme3의 경우 XML_MSG의 크기는 150MB 정도인 반면에 Scheme3은 440MB~1,600MB(8개 색인필드의 경우)를 사용하므로 더 많은 시간을 소모한다. 더욱이 Q2 질의를 여러 번 수행하는 웜버퍼(warm buffer)의 경우, Scheme1과 Scheme3은 버퍼 캐쉬(cache)의 효과를 볼 수 있으나 Scheme2는 색인필드 2개를 사용한 경우 이외에는 캐쉬 효과가 미미하였다.

그림 15는 Q3질의에 대한 수행시간을 비교한 것이다. Scheme1의 경우에는 검색조건으로 사용된 두 개의 색인필드 모두에 대해 정의된 인덱스를 사용하여 질의를 처리하므로 1초 정도의 시간만을 소비하는 가장 좋은 성능을 나타내었다. Scheme2는 두 개의 색인필드 중 하나에 대해서 인덱스를 사용하고 그 검색결과에 대해

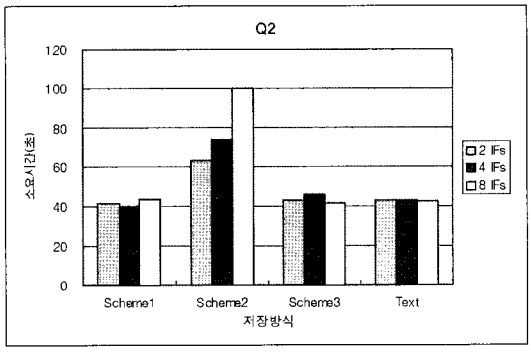


그림 14 Q2 질의수행 성능 비교

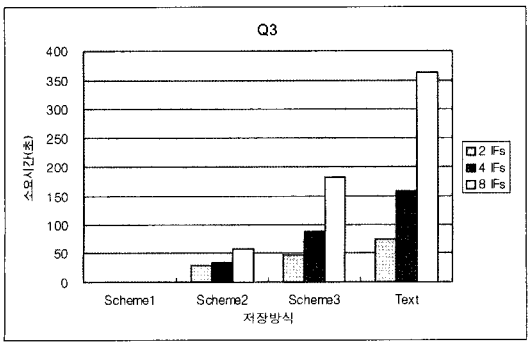


그림 15 Q3 질의처리 성능비교

나머지 색인필드의 조건을 검사하는 방식으로 수행되므로 Scheme1에 비해서는 성능이 떨어진다. Scheme3와 Text의 경우에는 색인필드에 대한 AND 조건이 XML_MSG 테이블과 INDEX_FIELD_VAL 테이블에 대한 중첩질의(nested sub-query) 형식으로 동작해야 하므로 질의처리에 따른 오버헤드가 가장 크고따라서 많은 시간을 소모한다.

5. 결 론

인터넷과 XML을 이용한 기업간통합시스템은 관련 기술의 표준화와 더불어 그 이용이 더욱 늘어날 전망이다. 본 논문에서는 기업간통합시스템의 XML 메시지 교환에 있어 반드시 필요한 기능인 메시지 저장시스템을 간단한 방식으로 구현하기 위한 저장구조를 제안하였다. 그리고 관계형데이타베이스를 사용하여 저장 구조를 구현할 수 있는 세가지 저장스키마를 제안하고 실험을 통해 각 방식의 디스크 공간 사용량과 질의처리 성능을 비교해 보았다.

실험 결과 저장스키마1이 대부분의 경우 저장공간의 사용량및 삽입, 검색 성능에 있어 가장 우수한 성능을 나타낸다. 저장스키마2는 인덱스를 사용하지 않았을 때에 삽입성능이 가장 우수하다. 따라서 저장스키마2는 저장된 메시지에 대해 조회가 거의 일어나지 않는 반면 메시지 보관이 중요한 경우에 적용 가능한 스키마로 판단된다. 저장스키마3의 경우에는 다른 것들에 비해 특별히 좋은 성능을 보이는 경우는 없다. 그러나 저장스키마3은 색인필드의 삽입과 삭제가 데이타베이스 스키마와는 상관없이 이루어 질 수 있다는 장점이 있다. 따라서 사용하는 관계형데이타베이스 시스템에서 테이블 컬럼의 추가 및 삭제를 지원하지 않는 경우에 유용하게 사용할 수 있는 스키마이다.

향후에는 XML 메시지로부터 색인필드의 값을 추출하는 과정에서 다수의 XPath를 사용하는 경우 이를 효율적으로 수행할 수 있는 방안에 대한 연구를 진행할 예정이다.

참 고 문 헌

[1] Christoph Bussler, "The Role of B2B Engines in B2B IntegrationArchitectures," SIGMOD Record, Vol. 31, No. 1, pp. 67-72, 2002.

[2] Brahim Medjahed, et. al., "Business-to-business interactions: issues and enabling technologies," VLDB Journal, Vol. 12, No. 1, pp. 59-85, 2003.

[3] Philip J. Harding, Quanzhong Li and Bongki Moon, "XISS/R: XML Indexing and Storage System using RDBMS," Proc. of International Conference on Very Large Databases, pp. 1073-1076, 2003.

[4] Harald Schoning, "Tamino-a DBMS Designed for

XML," Proc. of International Conference on Data Engineering, pp. 149-154, 2002.
[5] Denise Draper, Alon Y. Halevy and Deniel S. Weld, "The Nimble XML Data Integration System," Proc. of International Conference on Data Engineering, pp. 155-160, 2001.
[6] Andreas Renner, "XML Data and Object Database: The Perfect Couple?," Proc. of International Conference on Data Engineering, pp. 143-148, 2002.
[7] H. V. Jagadish, et. al., "TIMBER: A native XML database," VLDB Journal, Vol. 11, No. 4, pp. 274-291, 2002.
[8] Igor Tatarinov, et. al, "Storing and querying ordered XML using a relational database system", Proc. of ACM SIGMOD, pp. 204-215, 2002.
[9] Feng Tian, David J. DeWitt, Jianjun Chen, and Chun Zhang, "The Design and Performance Evaluation of Alternative XML Storage Strategies," ACM SIGMOD Record, Vol. 31, No. 1, pp. 5-10, 2002.
[10] Daniel Florescu and Donald Kossmann, "Storing and Querying XML Data using an RDBMS," Bulletin of the Technical Committee on Data Engineering, Vol. 22, No.3, pp. 27-34, 1999.

송 하 주
1993년 서울대학교 컴퓨터공학과 졸업(학사). 1995년 서울대학교 컴퓨터공학과 졸업(석사). 2001년 서울대학교 컴퓨터공학과 졸업(박사). 2003년 (주)아이티포웹 부장. 2003년 9월~현재 부경대학교 전자컴퓨터정보통신공학부 조교수. 관심분야는 데이타베이스, 웹서비스

김 창 수
1991년 중앙대학교 컴퓨터공학과 졸업(박사). 2002년~2003년 미국 UMKC 방문교수. 2004년~현재 한국멀티미디어학회 총무이사. 2004년~현재 부경대학교 대학원 부원장. 1992년~현재 부경대학교 전자컴퓨터정보통신공학부 교수. 관심분야는 Embedded System, 방재시스템, LBS/GIS

권 오 흠
1988년 서울대학교 컴퓨터공학과 졸업(학사). 1991년 KAIST 전산학과 졸업(석사). 1996년 KAIST 전산학과 졸업(박사). 1997년 한국전자통신연구소 연구원. 1997년 3월~현재 부경대학교 전자컴퓨터정보통신공학부 부교수. 관심분야는 알고리즘, 그래프이론