OWL 데이타 검색을 위한 효율적인 저장 스키마 구축 및 질의 처리 기법

(An Efficient Storage Schema Construction and Retrieval Technique for Querying OWL Data)

우 은 미 [†] 박 명 제 [††] 정 진 완 [†††]
(Woo, Eun-mi) (Park, Myung-Jae) (Chung, Chin-Wan)

요 약 현재 웹의 한계를 극복하기 위해 제안된 시맨틱 웹을 구축하기 위해서는 데이타에 잘 정의된 의미를 부여하는 온톨로지 언어를 사용해야 한다. W3C에서 제안한 OWL은 대표적인 온톨로지 언어이다. 시맨틱 웹 상에서 OWL 데이타를 효율적으로 검색하기 위해서는 잘 구성되어진 저장 스키마를 구축해야 한다. 본 논문에서는 효율적인 질의 처리를 위한 저장 스키마와 그에 적절한 질의 처리 기법을 제안하고자 한다. 또한 OWL 데이타는 클래스와 프로퍼티들의 상속 관계 정보를 포함한다. 따라서 질의 수행 시 질의에서 나타나는 클래스와 프로퍼티들 뿐 아니라 그것들과 관련된 계층 구조에 대한 탐색이 필요하다. 본 논문은 계층 정보를 유지하는 XML 문서를 생성하여 XML 데이타베이스 시스템에 저장한다. 이때 부모/자식 관계 추출에 용이한 기존의 넘버링 기법을 기반으로 노드의 순서 정보를 XML 문서의 애트리뷰트로 유지함으로써 질의에서 나타나는 클래스와 프로퍼티의 하위 정보들을 효율적으로 추출하고자 한다. 마지막으로 실험을 통한 질의 처리 성능의 비교를 통해서 본 논문에서 제안하고자 하는 기법들이 효과적임을 보인다.

키워드 : 시맨틱 웹, 온톨로지, OWL, OWL-QL

Abstract *With respect to the Semantic Web proposed to overcome the limitation of the Web, OWL has been recommended as the ontology language used to give a well-defined meaning to diverse data. OWL is the representative ontology language suggested by W3C. An efficient retrieval of OWL data requires a well-constructed storage schema. In this paper, we propose a storage schema construction technique which supports more efficient query processing. A retrieval technique corresponding to the proposed storage schema is also introduced. OWL data includes inheritance information of classes and properties. When OWL data is extracted, hierarchy information should be considered. For this reason, an additional XML document is created to preserve hierarchy information and stored in an XML database system. An existing numbering scheme is utilized to extract ancestor/descendent relationships, and order information of nodes is added as attribute values of elements in an XML document. Thus, it is possible to retrieve subclasses and subproperties fast and easily. The improved query performance from experiments shows the effectiveness of the proposed storage schema construction and retrieval method.*

Key words : Semantic Web, Ontology, OWL, OWL-QL

1. 서 론

월드 와이드 웹(World Wide Web)은 놀랄만한 속도로 발전을 거듭하고 있다. 하지만 사용자와 공유하는 정보의 양이 점점 증가함에 따라서 사용자가 원하는 정확한 정보를 얻을 수 없다는 문제점이 제기되고 있다. 이러한 가장 큰 원인은 현재의 웹이 사람이 읽고 해석하기 편리하도록 구성되어 있다는 데에 있다. 따라서 이런 웹의 문제점을 해결하고자 시맨틱 웹(Semantic Web)이

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성
· 지원 사업(IITA-2006-C1090-0603-0031)의 연구결과로 수행되었음
† 정 회 원 : 한국과학기술원 전산학과
emwoo@islab.kaist.ac.kr
†† 학생회원 : 한국과학기술원 전산학과
jpark@islab.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
chungcw@islab.kaist.ac.kr
논문접수 : 2005년 10월 4일
심사완료 : 2007년 3월 18일

제안되었다.

시맨틱 웹은 웹 상의 정보에 관련 의미를 부여해서 사람 뿐 아니라 컴퓨터가 정보의 의미를 분석하는 것을 가능하게 한다[1]. 이러한 시맨틱 웹을 구축하기 위해서는 웹 상에 존재하는 정보의 의미를 형식적으로 기술할 수 있는 온톨로지(Ontology) 언어를 사용해야 한다. RDF[2]는 W3C의 가장 기본적 시맨틱 웹 언어로서 웹에 있는 자원에 관한 메타정보를 표현하기 위한 언어이다. 온톨로지 언어인 DAML+OIL[3]은 RDF를 기반으로 표현되며, DAML+OIL의 언어적 특성을 계승하고 발전시킨 OWL[4]은 W3C의 권고안으로 채택된 대표적인 온톨로지 언어이다. OWL은 단지 사람에게 정보를 표시하는데 그치지 않고 정보의 내용을 직접 처리할 수 있는 어플리케이션을 구현하는데 활용될 수 있도록 설계된 언어이다. 이러한 OWL은 풍부한 어휘(vocabulary)와 형식적 의미론(formal semantics)을 포함하고 있기 때문에 기계 해석이 가능한 웹 컨텐트를 제작하는데 있어서 XML, RDF, 및 RDF 스키마보다 뛰어나다.

OWL 관련 연구는 현재 진행 중에 있거나 시작 단계에 있는 경우가 많으나, OWL 언어 이전에 만들어진 DAML+OIL을 지원하는 시스템으로는 DLDB[5]가 대표적인 예이며 RDF 저장/검색 시스템인 Sesame[6]을 확장시켜 만든 DAML+OIL 추론기(Reasoner)를 포함한 BOR[7]도 여기에 해당한다. OWL이 이전 온톨로지 언어에 비해 달라진 점은 정보 표현 범위가 넓어지고 추론 기능이 강화되었다는 것이다. 폭넓은 추론 기능을 지원하기 위해서는 OWL 시스템에 추론기가 포함되어 있어야 한다. 이미 OWL을 포함한 온톨로지 언어의 추론 기능을 지원하는 몇몇 추론 엔진이 개발되어 있다. 대표적인 추론기로는 Racer[8]와 Pellet[9]이 있다.

시맨틱 웹 상에서 정보를 검색하기 위해서는 OWL 데이타를 효율적으로 저장하고 검색하는 방법이 필요하다. 따라서 본 논문에서는 시맨틱 웹을 구축하기 위한 온톨로지 언어인 OWL 데이타를 저장, 검색하는 기법을 제안하고자 한다. 특히 OWL 데이타를 저장하기 위한 저장 스키마를 구축하는 데에 있어서 효율적인 질의 처리를 할 수 있는 기반이 되어질 저장 스키마를 제안한다. 그리고 OWL 데이타 질의 처리 기법에 있어서도 질의 처리 성능을 보다 향상시키는 기법을 제안하고자 한다.

OWL 데이타는 XML[10]과 유사한 형태로 표현된다. 계층 구조를 가지는 XML 데이타와 마찬가지로 OWL 데이타도 클래스(class)나 프로퍼티(property)에 대한 계층 구조를 가지며 질의 시 클래스나 프로퍼티의 하위 구조까지 고려해서 데이타를 검색하는 것이 필요하다. 본 논문에서는 대부분의 온톨로지 데이타를 RDBMS에 저장하되, 효율적으로 계층 정보를 추출하기 위해서 계층 정보의 저장은 별도로 XML 데이타베이스 시스템을 사용한다. 또한 XML 데이타를 위해 제안된 넘버링(numbering) 기법을 사용하여 보다 빠르고 효율적으로 질의 처리시 하위 구조를 고려할 수 있도록 하는 방법을 소개하고자 한다.

다양한 질의를 바탕으로 한 실험을 통해서 본 논문에서 제안한 저장 스키마와 질의 처리 기법이 효율적임을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 OWL 저장, 검색 시스템에 관한 관련 연구를, 3장에서는 OWL 시스템의 새로운 접근 방법을 소개한다. 4장에서는 논문에서 제안하는 저장 기법을, 5장에서는 질의 처리 기법을 설명한다. 6장에서는 실험 환경 및 실험 결과를 제시하고, 7장에서는 결론을 내린다.

2. 관련 연구

2.1 DLDB

DLDB[5]는 Lehigh University의 Semantic Web & Agent Technology Lab에서 개발한 시스템으로 RDBMS를 사용하여 온톨로지 데이타를 저장하는 시스템이다. DLDB는 DAML+OIL을 지원하기 위해 고안되었고, RDBMS로는 MS Access를 사용한다.

DLDB는 문서 상에 존재하는 클래스나 프로퍼티에 대해서 모두 테이블을 생성해 주는 저장 스키마를 가진다. 클래스 테이블에는 해당 클래스의 인스턴스(instance)들을 저장하고 각 인스턴스가 가지는 프로퍼티 값에 대한 정보들은 해당 프로퍼티 테이블을 사용해서 저장한다. 또한 클래스나 프로퍼티에 대한 계층 정보를 추출 하기 위해서는 별도로 계층 정보를 저장하는 테이블을 유지한다.

그림 1은 DLDB의 저장 테이블에 대한 구조를 보여준다.

<"classname"_"currentSeqNum"_TABLE>과 <"propname"_"currentSeqNum"_TABLE>은 문서 상에 존재하는 클래스와 프로퍼티 각각에 대해서 생성되는 테이블들이다. ID 애트리뷰트(attribute)에는 클래스 인스턴스의 고유한 ID가 저장되며 SUBJECT 애트리뷰트에는 해당 프로퍼티를 가지는 클래스 인스턴스의 ID

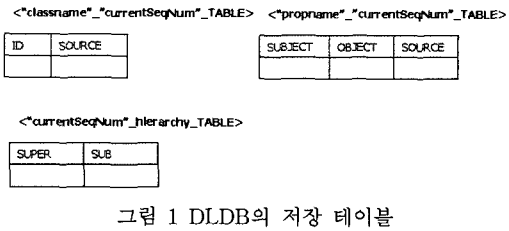

그림 1 DLDB의 저장 테이블

가, OBJECT 애트리뷰트에는 프로퍼티의 값이 저장된다. SOURCE 애트리뷰트에는 클래스나 프로퍼티가 속해 있는 온톨로지에 관한 정보가 저장된다. 각 테이블의 이름에는 currentSeqNum이 함께 붙는다. 이 숫자는 어떤 온톨로지가 데이타베이스에 처음 저장될 때마다 고유하게, 시스템 임의로 붙여지는 숫자이다.

<"currentSeqNum"_hierarchy_TABLE>은 각 DAML+OIL 데이타가 갖는 계층 구조를 저장하기 위한 테이블이다. 클래스/프로퍼티의 이름(SUB)과 해당 클래스/프로퍼티의 상위 클래스/프로퍼티의 이름(SUPER)을 저장한다.

특정 클래스나 프로퍼티에 대한 계층 구조를 추출하기 위해서는 <"currentSeqNum"_hierarchy_TABLE>에서 특정 클래스나 프로퍼트의 이름을 SUPER 애트리뷰트 값으로 포함하는 튜플을 찾아야 한다. 그리고 그 튜플의SUB 애트리뷰트를 다시 부모로 갖는 애트리뷰트를 찾는 작업을 반복해야 한다. 이와 같은 방법을 반복 적용하여 특정 클래스나 프로퍼티의 하위 클래스나 프로퍼티를 찾아가다가 더 이상의 하위 정보가 존재하지 않으면 탐색을 중단한다.

DLDB에서의 계층 구조 추출 방법은 계층 구조가 깊어질수록 하위 구조를 찾아가는 루틴이 계속 반복되므로 비효율적이다. 따라서 특정 클래스나 프로퍼티가 가지는 하위 구조가 적을 때는 성능에 영향을 거의 미치지 않으나, 하위 구조가 많을수록 계층 구조 추출에 많은 시간이 소요된다.

선언된 모든 클래스와 프로퍼티 각각에 대해서 테이블을 생성하는 것 또한 질의 처리에 있어서 성능을 저하시키는 요인으로 작용한다. 한 테이블 내에서 원하는 데이타를 찾기 위한 탐색 시간은 줄일 수 있으나 테이블의 수가 불필요하게 많아짐에 따라 테이블 사이의 많은 조인으로 인한 오버헤드(overhead)가 발생한다.

본 논문은 더 효율적인 계층 구조 추출을 위해 XML 문서 형태로 계층 구조 정보를 유지하였다. 따라서 계층 구조는 DLDB와는 달리단 한번의 테이블 탐색으로 계층 구조를 추출해 내는 것이 가능하며 DLDB의 계층 구조 추출에 비해 더욱 효율적이다. 또한 테이블 수를 적게 하여 조인의 오버헤드를 줄인다.

2.2 Sesame

Sesame[6]은 유럽의 IST 프로젝트인 On-To-Knowledge의 일부로서 Aidministrator Nederland b.v.에서 개발한 시스템으로 RDF와 RDF 스키마를 지원하는 온톨로지 데이타 저장, 검색 시스템이다. 본 논문에서 지원하는 OWL 표준 자체가 RDF를 기반으로 하기 때문에 Sesame의 저장 스키마를 비교, 검토 해 볼 필요가 있다.

Sesame에서는 문서 상에 존재하는 클래스와 프로퍼티에 대한 정보를 저장하기 위해 각각의 테이블을 사용한다. 또한 클래스의 인스턴스에 대한 정보를 저장하기 위해 하나의 테이블을 유지하며, 각 인스턴스가 가지는 프로퍼티 값에 대한 정보는 별도의 테이블에 저장한다.

그림 2는 Sesame의 저장 테이블에 대한 구조를 보여준다.

<class_TABLE>은 문서상에 존재하는 클래스들에 대한 정보를 담고 있는 테이블이다. 상위 클래스이건, 하위 클래스이건 관계없이 존재하는 모든 클래스들에 대한 이름을 저장한다.

<property_TABLE>은 <class_TABLE>과 비슷하게 문서 상에 존재하는 프로퍼티들에 대한 정보를 가지는 테이블이다.

<instanceof_TABLE>은 클래스의 인스턴스에 대한 정보를 유지하기 위한 테이블이다. CLASS 애트리뷰트에는 클래스의 이름이, INST 애트리뷰트에는 해당 클래스의 인스턴스 정보를 저장한다.

클래스의 인스턴스가 가지는 프로퍼티와 프로퍼티의 값에 대한 정보는 <triple_TABLE>에 저장된다. SUBJECT 애트리뷰트는 클래스 인스턴스, PREDICATE와 OBJECT 애트리뷰트에는 각각 클래스 인스턴스가 가지는 프로퍼티와 그 프로퍼티의 값이 저장된다.

클래스와 프로퍼티의 계층 정보를 위해서는 <subclassof_TLBLE>과 <subpropertyof_TABLE>을 유지한다. SUPER 애트리뷰트에는 부모 클래스/프로퍼티가,

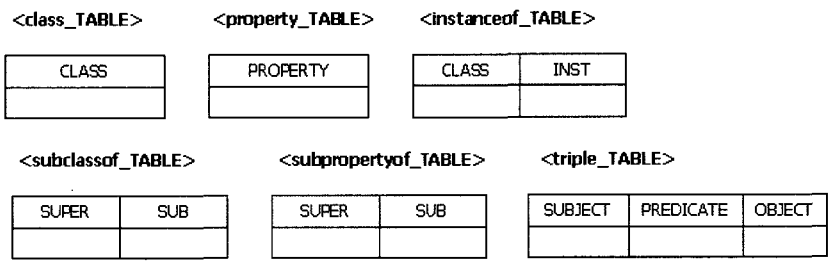

그림 2 - Sesame의 저장 테이블

SUB 애트리뷰트에는 자식 클래스/프로퍼티의 정보를 저장한다.

Sesame에서 사용하는 계층 정보 추출 방법은 DLDB에서의 방법과 유사하며 앞의 2.1절에서 설명한 것처럼 이러한 계층 정보 추출 방법은 상당히 비효율적이다. 특정클래스나 프로퍼티에 대한 계층 정보를 추출하기 위해서는 <subclassof_TABLE>이나 <subpropertyof_TABLE>에서 하위 클래스/프로퍼티를 찾는 작업을 반복해야 하므로 계층 구조가 깊어질수록 탐색 시간이 길어질 수밖에 없다.

또한 하나의 인스턴스 테이블에 모든 클래스의 인스턴스들을 저장하므로 테이블의 사이즈가 커질 수 밖에 없으며 이는 데이타 탐색 시 성능 저하 요인으로 작용할 수 있다.

3. 새로운 접근 방법

OWL을 지원하는 OWL 저장, 검색 시스템은 DAML+OIL을 지원하는 DLDB와는 근본적으로 다르다. 하지만 온톨로지 데이타를 저장하는 시스템으로서 DLDB와 유사한 점이 있다고 볼 수 있다. 앞 절에서 언급한 DLDB의 구조상의 단점들을 극복하기 위해서 RDBMS와 XML 데이타베이스 시스템을 함께 사용하는 방법을 고려할 수 있다. 따라서 이번 장에서는 OWL 데이타의 효율적인 저장을 위한 새로운 접근 방법을 소개하고자 한다.

클래스와 프로퍼티에 대한 계층 정보를 유지하기 위해서 XML 문서를 만들고 XML 데이타베이스 시스템에 저장한다. 즉, OWL 문서의 스키마 부분에 선언되어 있는 클래스와 프로퍼티 정보들 중에 상속 관계를 가지는 클래스와 프로퍼티의 고유한 ID를 가지고 간단한 XML 문서를 만든다.

그림 3은 OWL-QL 질의 시 직접적으로 접근해서 정보를 추출해야 하는 테이블 구조를 보여준다. 모든 클래스의 인스턴스와 해당 프로퍼티 그리고 프로퍼티 값에 대한 정보를 하나의 <OWL_Individual_TABLE>에 저장한다. 해당 클래스의 인스턴스(UID), 클래스 인스턴스가 속하는 클래스의 UID(TYPE), 클래스 인스턴스가 가지는 프로퍼티의 UID(Property), 그리고 클래스 인스턴스가 가지는 프로퍼티의 값(VALUE_U, VALUE_S)을 저장한다.

OWL-QL 질의 패턴(Query Pattern)마다 각각의 SQL문으로 변환을 시키고, 생성된 임시 테이블들을 마지막 SQL문을 통해서 모두 조인하는 방법을 취한다. SQL문으로 변환 시 하위 클래스/프로퍼티까지 검색을 하기 위해서는 질의문의 조건절에 계층 정보를 추가해 주어야 한다. 우선 XML 데이타베이스 시스템에 XPath [11] 질의를 사용하여 하위 구조를 추출해 낸다. 그리고

<OWL_Individual_TABLE>

UID	TYPE	Property	VALUE_U	VALUE_S

그림 3 OWL 저장, 검색시스템의 저장 스키마

그 하위 정보들은 생성된 SQL문의 WHERE절에 추가시킨다.

이러한 접근 방법은 XML 문서의 특성을 이용하여 계층 정보의 추출을 좀 더 효율적으로 가능하게 한다. 하지만 추출된 계층 정보를 기반으로 하위 클래스의 인스턴스나 프로퍼티의 값을 테이블에서 찾을 때 해당되는 값인지 일일이 비교해 보아야 하는 번거로움이 따른다. 또한 하나의 인스턴스 테이블을 사용하므로 테이블에 저장되는 데이타의 양이 늘어남에 따라서 테이블의 검색 시간 또한 증가한다는 단점이 있다.

본 논문은 이번 장에서 소개한 OWL 저장, 검색의 접근 방법을 좀 더 효율적으로 개선하는 방안을 제안하고자 한다. 계층 정보 추출에 있어서는 이번 장에서 소개한 접근 방법과 같이 XML 문서를 생성하나, 본 논문에서는 넘버링 기법을 접목시켜 더 효율적인 추출을 가능하게 하였다. 또한 시스템의 질의 처리 성능을 보다 향상시키기 위해 효율적인 저장 스키마의 구조와 검색 기법을 제안한 데에 그 차이를 두고 있으며, 그 차이를 중심으로 본 논문의 저장 및 질의 처리 기법을 설명하고자 한다.

4. OWL 데이타의 저장 기법

4.1 저장 처리기 구조

그림 4는 OWL 데이타를 저장하기 위한 저장 처리기 구조를 보여준다.

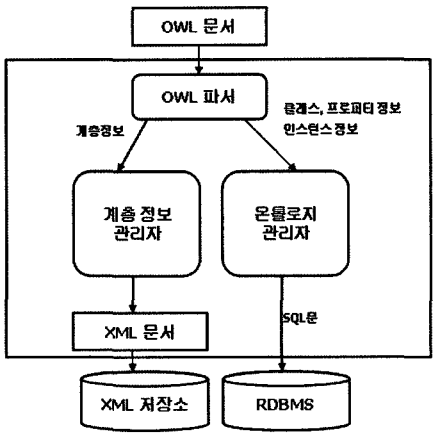

그림 4 저장 처리기 구조

OWL 문서가 입력으로 들어오면 OWL 파서에서 문서를 파싱해서 클래스, 프로퍼티 그리고 인스턴스에 관한 정보와 계층 정보, 프로퍼티에 대한 제약조건 등을 추출해 낸다. 이때 클래스와 프로퍼티에 대한 계층 정보는 계층 정보 관리자에 의해 XML 문서로 변환한다.

OWL 데이타의 계층 정보를 포함한 XML 문서는 XML 저장소에 저장된다. OWL 파서를 통해서 추출된 나머지 정보들은 온톨로지 관리자를 통해서 RDBMS에 저장된다.

4.2 클래스와 프로퍼티에 대한 계층 정보의 저장

OWL 데이타는 클래스와 프로퍼티들의 상속 관계 정보를 포함한다. OWL 데이타에 대한 질의 처리 과정은 질의에서 나타나고 있는 클래스와 프로퍼티들 뿐 아니라 그것들과 관련된 계층 구조를 탐색하고 관계가 있는 모든 클래스와 프로퍼티들간의 관계에 대한 탐색 또한 필요하다. 따라서 OWL 데이타의 효율적인 검색을 위해서는 클래스와 프로퍼티의 계층 정보를 유지하는 것이 중요하다. XML 문서는 XML 질의 언어를 통해서 특정 엘리먼트(element)의 하위 구조를 쉽게 추출한다. 그러므로 본 논문에서는 OWL 문서 내의 클래스와 프로퍼티의 계층 정보를 포함하는 간단한 XML 문서를 생성하여 저장함으로써 질의 시 계층 정보의 추출을 효과적으로 수행한다. 이때 Dietz's Numbering[12] 기법을 사용해서 관련 연구의 OWL 저장, 검색시스템에서 사용하는 XML 문서에 비해 계층 정보 추출의 효율성을 높였다.

이 넘버링은 각 엘리먼트의 전위(preorder)와 후위(postorder) 정보를 가지고 조상/자손(ancestor/descendant)간의 관계를 쉽게 알 수 있다. 그림 5는 XML 문서를 트리 형태로 표현했을 때, 각 엘리먼트의 전위와 후위의 값을 해당 노드에 함께 나타낸 것이다. 예를 들어 (2,4)인 노드의 하위 노드들을 찾고자 할 때 preorder>=2이고 postorder<=4인 노드를 찾으면 (2,4)의 자손 노드들을 모두 추출해 낼 수 있다.

그림 6은 XML 트리 형태로 표현된 클래스와 프로퍼티의 계층 정보를 바탕으로 XML 문서를 생성한 것이

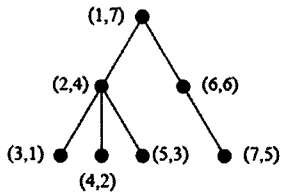

그림 5 Dietz's Numbering의 예

다. XML 트리에서 각각의 노드 안의 숫자는 클래스와 프로퍼티가 갖는 고유한 UID이며 노드 옆에는 각 노드의 클래스 또는 프로퍼티 이름과 (preorder, postorder)의 정보를 표기하였다. XML 문서의 각 엘리먼트는 order라는 애트리뷰트를 가지며 최상위 클래스나 프로퍼티의 이름과 함께 전위, 후위 정보를 값으로 가진다. 오른쪽의 XML 문서에서 C23 엘리먼트의 경우를 보면, UID가 23인 클래스의 최상위 클래스는 Liquor이고 전위의 값은 0, 그리고 후위의 값은 2임을 알 수 있다.

이때 어떤 클래스나 프로퍼티가 최상위 클래스나 프로퍼티이면서 하위 클래스나 프로퍼티를 가지지 않는 경우, 즉 (preorder, postorder)의 값이 (0,0)인 경우는 XML 문서에 포함하지 않는다. 그림 6에서 locatedIn, hasFlavor 프로퍼티가 이 경우에 해당한다.

4.3 저장 스키마

OWL 데이타는 크게 스키마 부분과 인스턴스 부분으로 구분된다. 스키마 부분은 클래스와 프로퍼티들간의 관계를 나타내고 있으며, 인스턴스 부분은 인스턴스가 속하는 클래스 그리고 프로퍼티와 프로퍼티 값이 포함된다.

그림 7은 클래스 인스턴스에 관한 정보를 저장하는 테이블들을 보여준다. 클래스 인스턴스와 그에 관련된 정보들을 저장하기 위한 테이블은 문서 내에 선언된 클래스나 프로퍼티에 따라 달라진다. 즉 선언된 클래스나 프로퍼티 각각에 대해서 테이블을 생성하게 된다. 만약, 클래스나 프로퍼티가 계층 구조를 가지는 경우에는 상위 클래스/프로퍼티, 하위 클래스/프로퍼티에 관한 테이

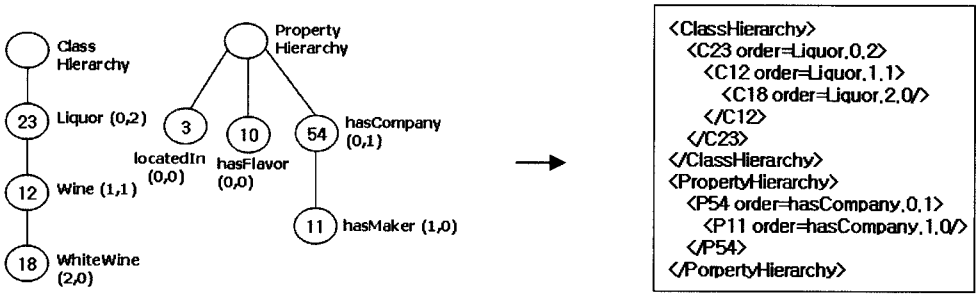

그림 6 계층 정보 저장을 위한 XML 문서 생성

<"classname"_TABLE>

Preorder	Postorder	UID

<"propertyname"_TABLE>

Preorder	Postorder	classUID	VALUE_U	VALUE_S

그림 7 클래스 인스턴스 정보를 위한 테이블

블들을 모두 생성하지 않고 최상위 클래스/프로퍼티에 대한 테이블만 생성한다. 하위 클래스/프로퍼티에 관한 정보들은 자신들의 최상위 클래스/프로퍼티 테이블에 저장된다.

앞서 설명한 그림 6 왼쪽의 XML 트리에 나타나 있는 클래스와 프로퍼티 정보를 가지고 테이블을 생성하면 Liquor 클래스, 그리고 locatedIn, hasFlavor, hasCompany 프로퍼티에 대해서만 테이블이 생성된다. Wine과 WhiteWine 클래스는 Liquor의 하위 클래스이므로 이 두 클래스에 대한 인스턴스들은 <Liquor_TABLE>에 저장된다. 마찬가지로 hasMaker 프로퍼티의 값은 최상위 프로퍼티 테이블인 <hasCompany_TABLE>에 저장된다.

각 테이블은 Preorder와 Postorder라는 애트리뷰트를 가진다. 최상위 클래스/프로퍼티 테이블에 모든 하위 클래스/프로퍼티의 내용을 가지므로 전위와 후위 정보를 가지고 특정 하위 클래스/프로퍼티를 손쉽게 찾을 수 있다. <"classname"_TABLE>의 UID는 해당 클래스 인스턴스의 고유한 ID이다. <"propertyname"_TABLE>의 classUID는 프로퍼티가 속하는 클래스의 고유 ID를 일컫는 것이며 VALUE_U나 VALUE_S는 프로퍼티가 가지는 값을 위한 애트리뷰트이다. VALUE_U 애트리뷰트에는 클래스 인스턴스가 가지는 프로퍼티의 값이 클래스 인스턴스인 경우에 그 인스턴스의 UID가 저장되며, 데이타 값인 경우에는 VALUE_S 애트리뷰트에 그 데이타 값이 저장된다.

하나의 인스턴스 테이블에 클래스의 인스턴스와 프로퍼티의 값들을 저장하지 않고 클래스나 프로퍼티에 따라서 테이블을 나누는 이유는 질의 시 검색 속도를 향상시키기 위해서이다. OWL 데이타에 대해서 하나의 인스턴스 테이블을 생성한다면 선언된 모든 클래스의 인스턴스들과 프로퍼티의 값들이 인스턴스 테이블 하나에 저장된다. 따라서 OWL 데이타의 양이 많아질수록 튜플의 수가 커지게 된다.

만약 클래스 A의 인스턴스들을 탐색하고자 할 때 인스턴스 테이블의 처음 튜플부터 마지막 튜플까지 검사하면서 클래스 A의 인스턴스들만을 추출해내야 한다. 인스턴스 테이블은 모든 클래스의 인스턴스들을 포함하므로 클래스 A의 인스턴스들을 포함한 튜플보다는, 클래스 A의 인스턴스들을 포함하지 않는 튜플이 더 많다. 따라서 특정 클래스의 인스턴스 정보를 탐색할 때 비효율적이다.

반면, 클래스와 프로퍼티에 대해 각각 나누어진 테이블로부터 원하는 데이타를 얻는 것은 용이하다. 클래스 A가 최상위 클래스라면 클래스 A에 대한 테이블로부터의 모든 튜플이 클래스 A의 인스턴스들을 포함한다. 만약 클래스 A가 최상위 클래스가 아니라 하더라도 클래스 A의 최상위 클래스에 대한 테이블에서만 검색하는 것이므로 하나의 인스턴스 테이블보다 훨씬 적은 수의 튜플만을 검사한다. 따라서 클래스와 프로퍼티에 대해서 테이블을 나눌 때 검색 시간이 더 적게 소요된다.

5. OWL-QL 질의 처리 기법

5.1 질의 처리기 구조

본 논문에서는 질의 처리를 위해서 OWL 데이타를 위한 질의 언어인 OWL-QL[13]을 사용한다. OWL-QL은 Stanford University의 Knowledge Systems Laboratory에서 만든 OWL 데이타의 질의 언어로 DAML[14] 질의 언어인 DQL[15]를 계승, 발전시킨 형태이다.

일반적으로, OWL-QL 질의는 검색하고자 하는 질의 내용을 포함하기 위한 질의 패턴 (Query Pattern), 질의가 수행되어야 하는 기반 온톨로지 데이타를 명시하기 위한 지식 베이스(Knowledge Base), 그리고 질의 패턴에 사용된 변수 리스트(Variable List)로 구성되어 있다. 또한 질의 결과를 원하는 형태에 맞도록 구성되도록 하기 위한 결과 패턴과 질의 결과의 수를 선택적으로 명시하도록 지원한다.

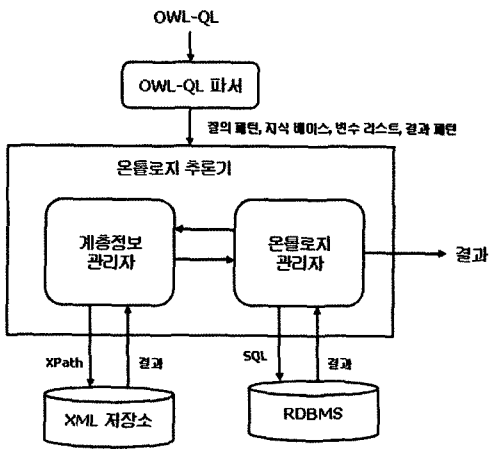

그림 8 질의 처리기 구조

본 논문에서 제안한 OWL-QL 질의 처리기의 구조는 그림 8에 나타나 있다. OWL-QL의 질의가 입력으로 들어오면 OWL-QL 파서는 질의를 파싱하여 질의 패턴, 지식 베이스, 변수 리스트, 결과 패턴에 대한 정보를 추출한다.

OWL 데이타의 클래스 및 프로퍼티의 계층 정보가 XML 문서 형태로 저장되어 있기 때문에, OWL-QL 질의 처리에 필요한 클래스 및 프로퍼티의 계층 정보의 추출을 위해서는 계층 정보의 저장에 사용된 XML 문서의 구조 정보에 따라 적절한 XPath 질의를 생성하는 작업이 필요하다. 따라서 계층 정보 관리자에서 적절한 XPath 질의를 만들고, XML 저장소로부터 계층 정보를 얻는다. 사용자가 원하는 최종 온톨로지 데이타를 얻기 위해서는 SQL 질의를 만들어서 RDBMS에 질의를 보내고 결과를 얻어야 하는데, 이때 전에 추출해 놓은 계층 정보를 SQL 질의의 where절에 추가시켜줌으로써 하위 클래스나 프로퍼티까지 고려를 해서 OWL-QL 질의에 대한 최종 결과를 얻는다.

5.2 SQL 질의로의 변환

5.2.1 OWL-QL 질의 패턴의 분류

OWL-QL 질의 패턴은 크게 4가지로 나누어 볼 수 있다. 그림 9는 각 특징에 따라 OWL-QL 질의 패턴을 분류해 놓은 예이다. 질의 패턴은 사용자가 원하는 결과가 클래스가 주어졌을 때의 클래스의 인스턴스인지(패턴 1), 프로퍼티가 주어졌을 경우의 클래스 인스턴스인지 (패턴 2), 프로퍼티의 값인지(패턴 3) 또는 클래스 인스턴스와 프로퍼티 값 두개인지 (패턴 4)에 따라서 4가지 패턴으로 나눌 수 있다.

그림 10은 OWL-QL 질의의 예이다. 각각의 질의 패턴은 해당 패턴으로 분류된다. 그림 10의 질의는 WhiteWine의 생산지(locatedIn)가 CentralTexasRegion 일 때의 생산자(hasMaker)와 향(hasFlavor)에 관한 질의이다.

```
패턴1: (type ?x WhiteWine)
패턴2: (locatedIn ?x Texas)
패턴3: (locatedIn StGenevieveTexasWhite ? y)
패턴4: (locatedIn ?x ?y)
```

그림 9 OWL-QL 질의 패턴의 분류

5.2.2 클래스와 프로퍼티에 대한 계층 정보의 추출

질의 패턴을 SQL문으로 변환하기에 앞서 제일 먼저 요구되는 것은 해당 클래스/프로퍼티의 하위 클래스/프로퍼티를 찾는 것이다. 하위 구조를 탐색하기 위해서는 앞의 4.2절에서 생성한 XML 문서를 사용한다. XML 문서는 클래스와 프로퍼티의 계층 정보를 유지하므로 손쉽게 계층 정보의 추출이 가능하며, 이때 질의를 위해서는 XPath를 사용한다. 계층 정보의 추출을 위한 XPath 질의는 단순하다. 질의 패턴에 나타나 있는 클래스나 프로퍼티에 해당하는 엘리먼트의 order 애트리뷰트 값을 추출할 수 있도록 XPath 질의문을 만들어 주면 된다. 그림 10의 OWL-QL 질의에서는 클래스 WhiteWine(C18)과 프로퍼티인 locatedIn(P3), hasMaker(P11), hasFlavor(P10)의 계층 정보를 추출해야 한다.

그림 11은 그림 10의 OWL-QL 질의에 나타나는 클래스와 프로퍼티의 계층 정보 추출을 위해 만들어지는 XPath와 그 결과의 예이다. 계층 정보 추출은 그림 6에서 만들어진 XML 문서를 기반으로 하였다.

XPath 질의를 실행하면 최상위 클래스/프로퍼티의 이름, 전위와 후위 정보를 결과로 얻을 수 있다. WhiteWine 클래스의 경우, 최상위 클래스가 Liquor이고 전위가 2, 후위가 0임을 알 수 있다. 최상위 프로퍼티이면서 동시에 자식 프로퍼티를 가지지 않는 locatedIn과 hasFlavor 프로퍼티의 경우, XML 문서의 엘리먼트로 존재하지 않으므로 XPath 질의를 수행해도 얻을 수 있는 결과가 없다. 이렇게 XPath 질의 수행 시 결과가 존재하지 않으면 최상위이면서, 자식이 없는 클래스나 프로퍼티로 간주한다.

그림 11에서 document ("Hierarchy.xml")// P3/@order의 XPath 질의를 수행 시 결과 값이 없다. 따라서 P3에 해당하는 프로퍼티인 locatedIn은 최상위 프로퍼티이며, 전위, 후위의 값은 각각 0이므로 locatedIn.0.0의 결과를 갖는 것으로 생각할 수 있다.

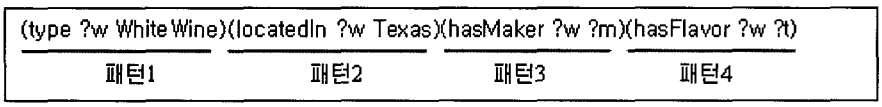

그림 10 OWL-QL 질의의 예

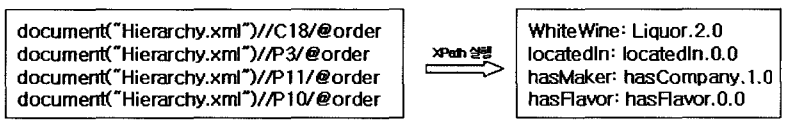

그림 11 계층 정보 추출시의 XPath 질의 예제와 그 결과

5.2.3 타입에 따른 SQL 변환

하나의 질의 패턴은 하나의 SQL문으로 변환된다. 각각의 SQL문에서 나온 결과는 마지막 SQL 구문에서 조인 시 사용한다. SQL문으로 변환 시 각 패턴에 따라서 변형된 SQL 변환 기법이 적용된다.

패턴 1~3까지의 질의 패턴은 하나의 변수를 가진 경우이다. 각각의 질의 패턴을 SQL문으로 변환하되, 만약 동일한 변수를 가진 질의 패턴이 두 개 이상 있을 경우 각각의 질의 패턴에 대한 SQL 문 사이에 INTERSECT 연산자를 사용한다. 그림 10의 OWL-QL 예제에서 첫 번째와 두 번째의 질의 패턴에 속해 있는 변수는 동일하다. 따라서 각각 SQL문으로 변환한 후에 INTERSECT 연산자를 적용하여 공통으로 가지지 않은 변수의 값을 제거해 준다. 이런 과정은 중간에 불필요한 튜플을 미리 제거함으로서 마지막 단계에서 조인되어야 할 튜플의 수를 줄인다.

패턴 4의 질의 패턴은 두 개의 변수를 가지고 있는 경우이며, 그 중 적어도 하나의 변수는 패턴 1~3의 질의 패턴에서 이미 사용된 것이다. 그림 10의 OWL-QL 예제에서 세 번째와 네 번째 질의 패턴의 ?w라는 변수는 첫 번째, 두 번째 질의 패턴에서 사용된 것이며, 이 두 질의 패턴의 SQL문으로부터 변수 ?w의 중간 결과의 값을 얻을 수 있다. 따라서 패턴 4의 질의 패턴에 대해서 SQL문으로 변환 시 변수 ?w의 중간 결과의 값과 매치가 되는 것만을 결과로 추출해 내도록 한다.

그림 12는 그림 10의 OWL-QL의 예를 가지고 SQL문으로 변환한 결과이다. 질의 패턴내의 클래스나 프로퍼티에 관한 계층 정보 추출의 결과는 그림 11을 참조하였다. 온톨로지데이타 추출은 XPath 질의의 결과로 얻은 최상위 클래스/프로퍼티에 관한 테이블로부터 이루어진다. 하위 구조까지 고려해서 튜플을 검색하기 위해서 각 SQL문의 WHERE절에 전위와 후위의 범위를 조건으로 주었으며, R2와 R3의 경우 이미 중간 결과를 얻은 변수 ?w의 값과 매치가 되는 것 중에서 데이타 추출을 위해 WHERE절의 조건으로 R11.w=CLASSUID가 추가되었다.

```
WITH
R11(w) AS (SELECT UID FROM Liquor_TABLE
           WHERE preorder>=0 AND postorder<=2
           INTERSECT
           SELECT CLASSUID FROM locatedIn_TABLE
           WHERE preorder>=0 AND postorder<=0 AND value_u=13)
R2(w, m) AS (SELECT CLASSUID, VALUE_U FROM hasCompany_TABLE
           WHERE preorder>=1 AND postorder<=0 AND R11.w=CLASSUID)
R3(w, f) AS (SELECT CLASSUID, VALUE_U FROM hasFlavor_TABLE
           WHERE preorder>=0 AND postorder<=0 AND R11.w=CLASSUID)

SELECT R11.w, R2.m, R3.f
FROM R11, R2, R3
WHERE R2.w=R3.w
```

그림 12 SQL 변환의 예

6. 성능 평가

본 논문에서 제안한 시스템과 3장에서 소개한 OWL 저장, 검색의 접근 방법과의 질의 처리 성능을 비교 분석하고자 한다.

6.1 실험 환경

실험에서 사용된 시스템의 사양은 펜티엄 4로CPU는 2.56GHz이고 메모리는 512MB이다. RDBMS는 DB2 데이타베이스 시스템을 사용하였으며, 자바를 사용하여 구현하였다.

실험에 사용된 데이타는 Univ-Bench Artificial data generator(UBA)[16]에 의해서 생성된 것이다. UBA는 앞서 관련 연구에서 설명한 DLDB 시스템을 개발한 Lehigh University의 연구팀이 만든 데이타 생성기로 OWL 데이타를 생성하는 것이 가능하다. 생성된 온톨로지 데이타는 대학교(university)와 학과(department) 그리고 대학교 내에서 이루어지는 활동(activity)에 관한 내용으로 이루어졌으며 스키마에는 43개의 클래스와 32개의 프로퍼티를 포함하고 있다. UBA를 사용하여 각각 5MB, 10MB, 15MB, 20MB, 25MB, 30MB, 40MB의 OWL 문서를 생성하였고 만들어진 7개의 문서를 기반으로 질의 성능을 측정하였다. OWL 데이타에 관한 질의는 UBA를 개발한 Lehigh University의 연구팀의 DAML+OIL에 관한 벤치마킹 논문[17]을 참조하였다. 실험을 위해서 11개의 질의를 생성하였으며 그 중 9개의 질의는 벤치마킹 논문을 참조하였고 나머지 두 개는 본 논문에서 제안한 계층 정보 추출에 관한 실험을 위해 생성하였다. 표 1은 실험에 사용된 질의의 특성에 관한 것이고, 표 2는 실험에 사용된 질의 리스트이다.

질의는 크게 5가지 특징으로 구분된다. 우선 하나의 클래스/프로퍼티를 가진 질의에 대해서 계층 구조를 가지지 않는 경우와 계층 구조를 가지는 경우로 구분한다. Q1은 전자의 경우에 해당하며 Q2, Q4, Q6는 후자의 경우에 해당된다. 둘째로 Q3, Q7은 하나의 클래스와 여러 개의 프로퍼티를 가진 경우이다. 그리고 여러 개의 클래스와 프로퍼티로 이루어진 복합 질의는 Q5와 Q8이

표 1 실험에 사용된 질의의 특징

Query Pattern TYPE		Query
하나의 클래스	계층구조 없음	Q1
하나의 프로퍼티	계층구조 있음	Q2, Q4, Q6
하나의 클래스 여러 개의 프로퍼티		Q3 Q7
여러 개의 클래스 여러 개의 프로퍼티		Q5 Q8
하나의 클래스		Q11
계층 정보 추출 테스트	계층구조 없음	Q9
	13레벨의 클래스 3레벨의 클래스	Q10

표 2 실험에 사용된 질의

	OWL-QL 질의 패턴
Q1	(type ?x GraduateStudent) (takesCourse ?x http://www.Department0.University0.edu/GraduateCourse0)
Q2	(type ?x Publication) (publicationAuthor ?x http://www.Department0.University0.edu/AssistantProfessor0)
Q3	(type ?x Professor) (worksFor ?x http://www.Department0.University0.edu) (name ?x ?y) (emailAddress ?x ?z) (telephone ?x ?t)
Q4	(type ?x Person) (memberOf ?x http://www.Department0.University0.edu)
Q5	(type ?x Student) (type ?y Course) (teacherOf http://www.Department0.University0.edu/AssociateProfessor0 ?y)
Q6	(type ?x Student) (takesCourse ?x http://www.Department0.University0.edu/GraduateCourse0)
Q7	(type ?x *UndergraduateStudent*) (takesCourse ?x http://www.Department0.University0.edu/Course25) (takesCourse ?x http://www.Department0.University0.edu/Course38) (name ?x ?y)
Q8	(type ?x GraudateStudent) (takesCourse ?x http://www.Department0.University0.edu/GraduateCourse16) (type ?y Department)(subOrganizationOf ?y http://www.University0.edu)
Q9	(type ?x FullProfessorXheadOf ?x http://www.Department0.University0.edu) (name ?x ?y)
Q10	(type ?x Employee)(memberOf ?x http://www.Department0.University0.edu) (name ?x ?y)
Q11	(type ?x Student)

해당한다. 네번째는 하나의 클래스에 대한 인스턴스들을 추출하는 단순 질의이다. Q11이 이 경우에 해당한다. 마지막으로 단말 노드에 해당하는 클래스/프로퍼티에 관한 질의와 하위 클래스, 프로퍼티를 많이 가지고 있는 상위 클래스/프로퍼티를 포함하는 질의를 수행하여 비교해 보았다. Q9와 Q10이 여기에 해당한다.

6.2 실험 결과

6.2.1 질의 성능

그림 13과 14는 각각 15MB와 25MB의 크기를 갖는 OWL 문서에 대해서 8개의 질의를 실행시킨 결과이다. method 1은 본 논문에서 제안한 기법을 기반으로 구현된 시스템이고 method 2는 3장에서 소개한 접근 방법에 관한 질의 성능 결과이다.

두 개의 그래프에서 보듯이 수행한 모든 질의에 대해서 본 논문에서 제안한 기법이 더 좋은 성능을 보이며 크게는 1.3배 정도의 차이를 보인다. 전반적으로 모든 질의에 대해서 성능이 좋게 나타나는 우선적인 요인은 저장 테이블을 클래스와 프로퍼티에 대해서 나누어 놓았다는 데에 있다. 비교 시스템의 경우, 어느 클래스나 프로퍼티를 접근하던지 간에 무조건 <OWL_Individual_TABLE>을 접근해서 결과를 얻는다. 하지만 본 논문에서 제안한 저장 스키마는 정의된 클래스와 프로퍼티에 따라서 나누어지기 때문에 테이블에 접근할 때에 더 적은 양의 데이타를 검색한다는 장점이 있다.

성능 결과를 자세히 살펴보면 Q3, Q5, Q7에 대한 성능이 특히 좋다. Q3과 Q7은 하나의 클래스에 대한 여러 개의 프로퍼티가 주어진 경우의 질의이고 Q5의 경우 여러 개의 클래스와 여러 개의 프로퍼티가 주어진 질의이다. 본 논문에서 제안한 질의 처리 기법은 OWL-QL 질의 패턴 중에서 하나의 변수만을 가진 질의 패턴들이 서로 같은 변수를 가졌다면 INTERSECT 연산자를 사용한다. 이런 과정에서 결과로 나올 수 없는 튜플들은 미리 제거가 된다. 그리고 패턴 4의 질의 패턴에 이미 중간 튜플들을 얻은 변수가 포함되어 있다면 패턴 4의 질의 패턴에 대한 SQL문을 만들 때 중간 튜플들의 값에 대해서 조인을 해주게 된다. 따라서 마지막 단계에서 조인되어야 할 튜플의 수를 줄일 수 있다.

하지만 비교 시스템의 경우, 질의 패턴 각각에 대해서 SQL문을 생성하고 마지막 단계에서 각각의 SQL문에 대해서 생성된 중간 결과들을 모두 조인한다. Q3의 (name ?x ?y)의 경우 name이라는 속성을 가지는 클래스 인스턴스와 그 속성값을 모두 검색 결과로 가져오고 마지막에 조인을 통해서 불필요한 튜플들을 제거해 준다. 따라서 중간 단계에서 많은 튜플들이 생성되어 조인해야 할 튜플의 수가 많아지게 되므로 본 논문에서 제안한 기법보다 성능이 저하된다. 그러므로 여러 개의 프로퍼티를 가진 경우, 특히 하나의 변수를 갖으면서 같은 변수를 가진 질의 패턴이 많은 경우에 본 논문에서 제

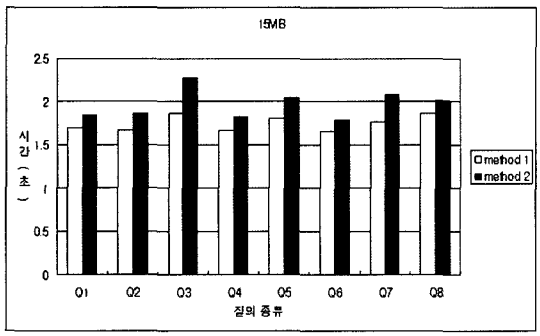

그림 13 15MB 문서에 대한 질의 성능

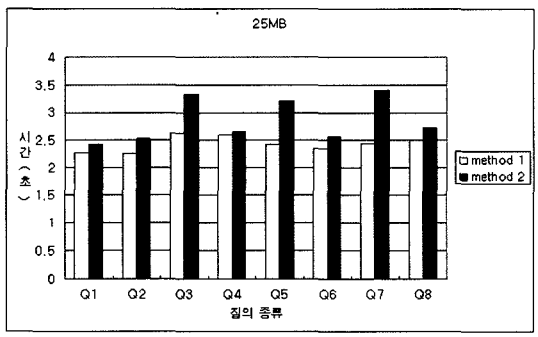

그림 14 25MB 문서에 대한 질의 성능

안한 질의 기법을 사용시 좋은 성능을 얻을 수 있다.

6.2.2 저장 스키마에 특히 영향을 받는 질의

그림 15는 Q11 즉 Student 클래스에 해당하는 모든 인스턴스들을 추출하는 질의에 대한 성능이 문서 크기가 커짐에 따라서 어떻게 변화되는가를 측정한 것이다. 저장 스키마의 영향만을 살펴보기 위해서 SQL 실행 시간만을 측정하였다. Q11은 단순히 클래스 인스턴스를 추출하는 질의이므로 SQL 변환 기법에 있어서는 본 논문과 비교 시스템간의 성능 차이가 없다. 따라서 수행 결과는 각 시스템의 저장 스키마에 좌우된다고 할 수 있다.

성능결과를 살펴보면, 본 논문에서 제안한 기법은 문서 크기가 늘어난다 하더라도 성능에 크게 영향을 받지 않는 것을 볼 수 있다. 하지만 비교 시스템은 문서의 크기가 늘어남에 따라 검색 시간 역시 증가한다. 앞서 설명한 것처럼 비교 시스템은 문서 크기가 커짐에 따라 <OWL_Individual_TABLE>의 크기가 계속적으로 증가하게 된다. Q11의 Student 클래스의 인스턴스를 추출하기 위해서도 테이블을 전부 검색해야 하므로 문서 크기가 커지면 실행시간도 증가할 수 밖에 없다.

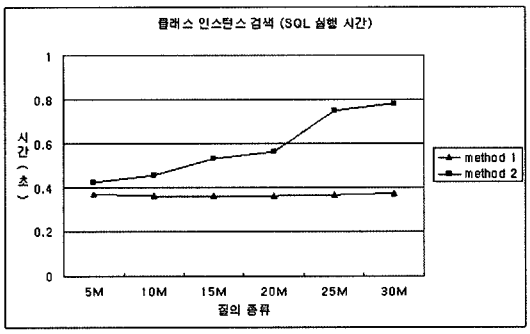

그림 15 하나의 클래스에 대한 인스턴스를 추출하는 질의

하지만 본 논문에서 제안한 저장 스키마는 클래스/프로퍼티마다 테이블을 분리했기 때문에 Student 클래스의 인스턴스를 추출하기 위해서는 최상위 클래스인 Student 클래스에 관한 테이블만 접근하면 된다. 따라서 문서 크기가 증가한다 하더라도 성능에 심한 영향을 받지 않는다.

6.2.3 하위 클래스/프로퍼티 추출시의 성능

마지막으로 계층 구조가 많은 경우와 적은 경우의 성능을 분석하고자 한다. 본 논문은 계층 정보 추출을 위해서 XML 데이타베이스 시스템을 이용하며, 저장 테이블에 Preorder와 Postorder 애트리뷰트를 두어 하위 구조를 손쉽게 추출할 수 있도록 제안하였다. Q9는 단말 노드의 클래스/프로퍼티에 관한 질의이고 Q10은 클래스와 프로퍼티가 많은 계층 구조를 포함하고 있는 경우이다. Q10의 질의에 있는 Employee 클래스는 13개의 하위 구조를 포함하며 memberOf 프로퍼티는 2개의 하위 구조를 포함한다. Q9 질의에 있는 FullProfessor는 Employee 클래스의 하위 클래스의 하나로 단말 노드이고 headOf 역시 memberOf 프로퍼티의 하위구조이다.

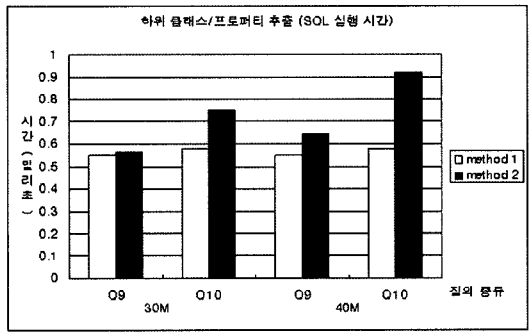

그림 16 하위 클래스/프로퍼티를 추출시의 성능 비교 질의

그림 16은 각각의 질의에 대한 SQL 실행 시간만을 측정한 것으로, 계층 구조에 따른 성능을 보여준다. 그림 16을 살펴보면 하위 구조를 갖지 않는 Q9 질의보다 하위 구조를 가진 질의 Q10에 대해서 비교 시스템보다 본 논문의 시스템이 더 나은 성능을 보이는 것을 알 수 있다. 또한 문서 크기가 커지면 성능 차이가 더욱 커진다. 비교 시스템의 경우 Employee 클래스의 하위 클래스를 고려하기 위해 SQL문에 "type=A OR type=B…" 라는 식의 OR 연산을 12번 사용해야 한다. 그리고 최악의 경우, 테이블의 각 튜플마다 13번의 조건에 대해 검사를 해 주어야 한다. 하지만 본 논문에서 제안하는 시스템은 전위와 후위의 범위를 고려하여 하위 클래스를 파악한다. 따라서 계층 구조를 많이 포함하면 할수록 본 논문의 기법이 유리하게 작용함을 알 수 있다.

7. 결 론

웹은 사용자와 정보량이 점점 증가함에 따라서 사용자가 원하는 정확한 정보를 추출하는 것이 어려워진다는 문제점이 대두되었다. 사람을 위해 디자인되어졌다는 현재의 웹의 문제점을 해결하고자 사람 뿐 아니라 기계까지도 정보의 의미를 해석해서 더 효율적인 질의 결과를 얻을 수 있는 시맨틱 웹이 대두되었고 현재 차세대 웹으로 각광을 받고 있다.

시맨틱 웹을 구축하기 위해서는 웹 문서에 포함된 정보의 의미를 형식적으로 기술할 수 있는 온톨로지 언어가 필요하다. 대표적인 온톨로지 언어로는 OWL이 있으

며 시맨틱 웹이 지속적으로 발전하기 위해서는 OWL 데이타를 효과적으로 저장, 검색할 수 있는 기법이 제안되어야 한다.

본 논문은 OWL 데이타에 질의 처리 성능을 향상시킬 수 있는 저장 스키마를 구축하는 방법과 그에 적절한 질의 처리 기법을 제안한다. 최상위 클래스/프로퍼티마다 테이블을 생성하고 하위 정보들은 해당되는 최상위 클래스/프로퍼티의 테이블에 저장하도록 하였다. 그리고 질의 처리 시 고려되어야 할 하위 클래스/프로퍼티를 추출하기 위한 계층 정보를 유지하기 위해서 XML 문서를 생성한다. 이때 XML 분야에서 제안된 넘버링 기법을 사용하여 효율적으로 해당 하위 구조를 추출하도록 하였다. 또한 질의 처리 시 사전에 불필요한 중간 튜플을 제거함으로써 마지막에 조인되어야 하는 튜플의 수를 줄였다.

실험을 통해서 수행한 모든 질의에 대해서 3장에서 소개한 새로운 접근 방법에 비해 본 논문에서 제안하는 시스템의 질의 처리 성능이 전반적으로 좋으며 약 1.3배의 성능 차이가 나는 것을 알 수 있다. 특히 여러 클래스와 여러 프로퍼티를 가지는 질의와 많은 계층 구조를 가지는 질의에 대해서 보다 나은 성능을 나타냄을 알 수 있다.

참 고 문 헌

[1] Tim Berners-Lee, James Hendler, and Ora Lassila, The Semantic Web, Scientific American, May 2001.

[2] Frank Manola, and Eric Miller, RDF Primer, W3C Recommendation, http://www.w3.org/TR/rdf-primer, 2004.

[3] Deborah L. McGuinness, Rechard Fikes, and James Hendler, DAML+OIL: An Ontology Language for the Semantic Web, IEEE INTERNET SYSTEMS, pp. 72-80, September/October 2002.

[4] Michael K. Smith, Chris Welty, and Deborah L. McGuinness, OWL Web Ontology Language Guide, W3C Proposed Recommendation, http://www.w3c.org/TR/ 2003/PR-owl-guide-20031215, 2003.

[5] Zhengziang Pau, and Jeff Heflin, DLDB: Extending Relational Databases to Support Semantic Web Queries, In Workshop on Practical and Scaleable Semantic Web System, ISWC, pp. 109-113, 2003.

[6] Jeen Broekstra, and Arjohn Kampman, Sesame: An Architecture for Storing and Querying RDF Data and Schema Information, Proceedings of the first International Semantic Web Conference(ISWC), 2002.

[7] Kiril Simov, and Stanislav Jordanov, BOR: a Pragmatic DAML+OIL Reasoner, IST Project IST-1999-10132 On-To-Knowledge.

[8] Volker Haarslev, and Ralf Moller, Description of the RACER System and its Application, University of Hamburg, Computer Science Department

[9] http:// www.mindswap.org/2003/pellet

[10] Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, Extensible Markup Language(XML) 1.0. W3C Recommendation, http://www.w3c.org/TR/REC-xml, 2004.

[11] James Clark, XML Path Language(XPath) Version 1.0 W3C Recommendation, http://www.w3c.org/TR/Xpath, 1999.

[12] Paul F. Dietz, Maintaining order in a linked list, Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, pages 122-127, May 1982.

[13] Richard Fkies, Patrick Hayes, and Ian Horrocks, OWL-QL - A Language for Deductive Query Answering on the Semantic Web

[14] http://www.daml.org/

[15] Richard Fikes, Pat Hayes, and Ian Horrocks, DAML Query Language, http://www.daml.org/2003/04/dql, 2003.

[16] http://www.lehigh.edu/yug2/Research/SemanticWeb/LUBM/LUBM.htm

[17] Yuanbo Guo, Jeff Heflin, and Zhengxiang Pan, Benchmarking DAML+OIL Repositories, The Semantic Web - ISWC 2003, pages 613-627, 2003.

우 은 미

2003년 충남대학교 컴퓨터공학과(학사)
2005년 한국과학기술원 전산학과(석사)
관심분야는 데이타베이스, 시멘틱 웹, OWL

박 명 제

정보과학회논문지 : 데이타베이스
제 34 권 제 2 호 참조

정 진 완

정보과학회논문지 : 데이타베이스
제 34 권 제 2 호 참조