Journal of The Korea Society of Computer and Information
Vol. 20, No. 3, March 2015
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2015.20.3.029

관계형 데이터베이스로부터 OWL 온톨로지를 추출하기 위한 SPARQL-DL 프로세서

최 지 웅*, 김 명 호*

SPARQL-DL Processor to Extract OWL Ontologies from Relational Databases

Ji-Woong Choi *, Myung-Ho Kim *

요 약

본 논문에서는 RDB로부터 가상적 변환에 의해 생성되는 OWL 온톨로지의 질의 응답을 위하여 OWL을 위한 질의어인 SPARQL-DL의 구현 방법을 제안한다. 제안하는 SPARQL-DL 프로세서는 입력된 SPARQL-DL 질의문을 내부에서 SQL 질의문으로 변환하여 실행시킨다. 이러한 질의 처리 방식은 두 가지의 장점이 있다. 첫째, RDB로부터 생성된 OWL 온톨로지를 저장하기 위한 별도의 저장소가 요구되지 않는다. 둘째, 대용량 ABox 추론에 문제점을 나타내는 Tableau 알고리즘 기반의 추론기의 사용 없이도 RDB 인스턴스로부터 생성된 대용량 ABox가 서비스 될 수 있다. 본 논문의 SPARQL-DL 질의문으로부터 SQL 질의문을 생성하는 알고리즘은 RDB와의 연결 수립에 따른 오버헤드를 최소화하기 위하여 입력된 하나의 SPARQL-DL 질의문이 하나의 SQL 질의문으로 변환되도록 설계되어 있다.

▸ Keywords : SPARQL-DL, SPARQL, 시맨틱 웹, OWL, 관계형 데이터베이스, RDF, 온톨로지

Abstract

This paper proposes an implementation of SPARQL-DL, which is a query language for OWL ontologies, for query-answering over the OWL ontologies virtually generated from existing RDBs. The proposed SPARQL-DL processor internally translates input SPARQL-DL queries into SQL queries and then executes the translated queries. There are two advantages in the query processing method. First, another repository to store OWL ontologies generated from RDBs is not required. Second, a large ABox generated from an RDB instance is able to be served without using Tableau algorithm based reasoners which have a problem

•제1저자 : 최지웅 •교신저자 : 김명호
•투고일 : 2014. 10. 15, 심사일 : 2014. 10. 23, 게재확정일 : 2014. 11. 6.
* 숭실대학교 IT대학 컴퓨터학부(School of Computer Science and Engineering, Soongsil University)
※ 이 논문은 2014년도 한국연구재단 기초연구사업(NRF-2014R1A1A2058695) 지원에 의해 연구되었음.

in large ABox reasoning. Our algorithm for query rewriting is designed to create one corresponding SQL query from one input SPARQL-DL query to minimize the overhead by establishing connections with RDBs.

▸ Keywords : SPARQL-DL, SPARQL, Semantic Web, OWL, Relational Database, RDF, Ontology

I. 서 론

W3C는 현재의 웹(WWW)을 문서의 웹으로 정의하며 시맨틱 웹을 데이터의 웹으로 정의한다. 현재의 웹은 표준 문서 포맷인 HTML과 표준 문서 교환 프로토콜인 HTTP를 통해서 웹 스케일의 문서 수준 데이터 통합을 이루었다. 이와 유사하게 시맨틱 웹은 표준 데이터 모델로서의 RDF와 OWL을 통해서 웹 스케일의 데이터 수준 데이터 통합을 하고자 한다. 즉, 시맨틱 웹의 비전은 웹 스케일의 표준화된 데이터베이스를 구축하고자 하는 것이다. 이에 현재의 웹 기저에서 데이터를 공급하는 데이터 원천들이 갖는 다양한 데이터 포맷들과 RDF 혹은 OWL 포맷과의 호환성을 확보하기 연구는 필연적이다. 연구 [1]에 의하면 동적 웹 페이지의 70% 이상이 관계형 데이터베이스(RDB) 데이터를 근간으로 생성된다. 따라서 관계형 모델에 기반한 RDB 데이터 및 데이터간의 관계를 RDF 혹은 OWL 모델로 표현하기 위한 연구는 시맨틱 웹 분야에서 가장 활발히 진행되고 있는 분야 중 하나이다.

RDB를 RDF 그래프 혹은 OWL 온톨로지로 변환하기 위한 방식은 물리적 변환과 가상적 변환이 있다. 물리적 변환은 RDB 전체 데이터 및 메타데이터(스키마)를 대상으로 ETL(Extract, Transform, and Load) 과정을 수행한다. 물리적 변환 결과인 RDF 그래프 혹은 OWL 온톨로지는 원본 DB와는 다른 별도의 저장소에 물리적으로 저장된다. 가상적 변환은 사용자에 의한 시맨틱 웹 질의 언어 형식의 질의 요청이 발생하는 시점마다 입력된 질의를 SQL 질의로 재작성한 후 원본 DB에 질의하며 그 질의 결과 셋 만을 변환하여 사용자에게 반환한다.

RDB를 물리적/가상적 변환에 의하여 생성된 RDF 그래프로서 서비스하기 연구는 상당히 성숙되어 있으며 이를 구현한 시스템 또한 상용 수준에 근접해 있다. 대표적인 연구 결과로서 D2RQ[2], Triplify[3], Virtuoso RDF Views[4]가 있다. W3C는 RDB의 RDF 그래프로의 변환과 관련한 문제들의 표준화를 위하여 2009년부터 RDB2RDF 워킹 그룹을 운영중이다[5].

OWL 온톨로지는 전형적으로 Tableau 알고리즘 기반의 추론기에 적재되어 서비스된다. Tableau 알고리즘은 TBox 추론에 최적화 되어 있으나 ABox 추론에 있어서는 다차시간에 다루지 못하는 문제가 있다.[6, 7]. 즉, RDB 인스턴스의 볼륨이 클 경우 ABox의 볼륨 또한 커지므로 이러한 온톨로지는 Tableau 추론기 상에서 서비스될 수 없음을 의미한다. 이러한 문제를 해결하기 위하여 Instance Store[6]와 LAS[7]와 같은 추론기는 TBox를 내부의 Tableau 추론기에 적재시키고 ABox를 RDBMS에 적재시키는 구조를 취한다. 이 추론기들은 각각 ABox를 저장하기 위한 고유의 DB 스키마를 가지고 있다. 따라서 이들 추론기를 사용하여 이미 구축되어 있는 RDB를 OWL 온톨로지로서 서비스하고자 할 때 원본 RDB를 이들 추론기 고유의 DB 스키마를 준수하는 DB로 변환해야 하므로 결국 ETL 과정이 수반된다. 즉, 이들 추론기의 사용은 RDB에서 OWL 온톨로지로의 물리적 변환이 발생함을 의미한다. Instance Store와 LAS와는 달리 Tableau 추론기의 사용을 완전히 배제한 KAON2[8] 추론기는 Tableau 추론기와 비교하여 처리 가능한 서술논리의 표현에 대한 제약이 존재하나 입력된 DL 계열 온톨로지를 내부적으로 disjunctive datalog program으로 변환하여 서비스함으로써 대량의 ABox를 처리할 수 있다. 그러나 KAON2 또한 RDB로부터 물리적 변환만이 가능하다.

본 논문의 SPARQL-DL 프로세서는 Instance Store와 LAS와 동일하게 TBox 요소를 수집하기 위해서 Tableau 추론기를 사용하며 ABox 요소를 수집하기 위해서 RDBMS를 사용한다. 그러나 Instance Store와 LAS와의 차이점은 ABox 저장을 위한 별도의 스키마를 요구하지 않는다. 즉, 기구축된 RDB는 그 자체로서 ABox 저장소 역할을 담당한다. 이것은 RDB 인스턴스로부터 ABox의 가상적 변환이 가능함을 의미한다.

RDB를 OWL로 번역한 온톨로지를 가상적 변환에 의하여 서비스하고자 할 경우 앞서 기술한 것처럼 기존의 추론기들에

의해서는 불가능하기 때문에, D2RQ, Triplify, Virtuoso RDF Views와 같은 시스템을 사용하여 RDF 그래프로서 서비스하고 있다. 이것은 W3C가 OWL이 RDF로서 해석될 수 있도록 설계하였기 때문이다. RDF 그래프는 일종의 edge labeled 방향 그래프 모델로서 RDF를 위한 질의어인 SPARQL 또한 그래프 패턴 매칭 방식이다. 즉, SPARQL 기반의 질의 응답은 그래프 패턴이 입력되면 패턴이 일치되는 부분 그래프들을 반환한다. 그러나 OWL은 논리 기반의 질의 응답을 지원할 수 있는 데이터 모델이기 때문에 그래프 패턴 매칭 방식보다 의미적으로 명료하며 효율적인 질의 응답이 가능하다. 따라서, 본 논문에서는 RDB로부터 가상적 변환에 의해 생성되는 OWL 온톨로지의 질의 응답을 위하여 OWL을 위한 질의어인 SPARQL-DL [9]을 구현한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구를 기술하며, 3장에서는 제안하는 시스템의 설계 내용을 기술한다. 4장에서는 제안하는 시스템의 주요 알고리즘에 대한 구현내용을 기술한다. 5장에서는 평가를 하며 6장은 결론에 해당한다.

II. 관련 연구

본 절은 RDB로부터 OWL 온톨로지를 생성하여 서비스하기 위한 기존 연구들에 대한 분석과 OWL-DL을 위한 질의어인 SPARQL-DL에 대한 기술 그리고 본 논문의 SPARQL-DL 프로세서가 추출하는 RDB로부터 변환된 OWL 온톨로지의 형상에 대한 기술로 구성된다.

1. RDB를 OWL 온톨로지로 제공하기 위한 연구

RDB를 OWL 온톨로지로 변환 및 서비스하기 위한 초기의 연구들은 RDB 스키마만을 변환 범위로 하여 TBox에 해당하는 OWL 온톨로지를 생성해 내는 것이 목적이었다. 연구 [10-12]는 이러한 범주의 대표적인 연구들이다. 이 연구들은 제 3 정규형을 만족시키는 스키마를 변환 대상으로 하는 것이 특징이다. 그러나 이들을 분석한 결과 스키마 내에 primary key 컬럼이 동시에 foreign key이고 컬럼의 개수가 3개 이상인 테이블이 존재할 경우 이러한 테이블을 변환하지 못하는 문제가 발견되었다. 제안하는 시스템에 내장된 RDB 스키마로부터 TBox를 생성하는 모듈은 이러한 문제를 해결한다.

RDBToOnto [13]과 SQL2OWL [14]는 RDB 스키마와 인스턴스 데이터를 입력으로 하여 TBox와 ABox로 구성된 OWL 온톨로지를 생성하기 위한 대표적인 연구들이다. 그러나 이들 연구는 물리적 변환에 의한 RDB 기반 OWL 온톨로지 생성에 적합하다. RDBToOnto는 온톨로지 생성을 위한 초기 단계로서 수작업이 개입되는 방식의 데이터베이스 정규화를 수행한다. 데이터베이스 정규화는 스키마 변경을 의미하므로 가상적 변환에 의한 온톨로지 서비스에는 적합하지 않다. SQL2OWL은 자동화된 프로세스에 의해 온톨로지를 생성한 후 마지막 단계에서 수작업으로 E-R(Entity-Relationship) 모델과 생성된 온톨로지를 대조하여 검증 및 정제하는 작업을 수행한다. 가상적 변환에 의한 서비스에서는 E-R 모델이 개입할 시점이 없기 때문에 이러한 방식은 물리적 변환에 의한 온톨로지 서비스에 보다 적합하다.

Ultrawrap [15]은 가상적 변환을 지원한다. 그러나 이 연구는 RDB 스키마는 OWL 온톨로지로 변환하지만 RDB 인스턴스 데이터는 RDF 트리플로 변환하는 것이 특징이다. RDB 스키마를 OWL로 변환하는 것은 RDF로 변환하는 것과 비교하여 관계형 모델의 의미를 보다 정확하게 반영하기 때문이다. RDB 인스턴스를 RDF로 변환하는 이유는 이 시스템이 사용자에게 제공하는 질의 인터페이스가 RDF를 위한 질의어인 SPARQL이기 때문이다. 현 시점에서 OWL을 위한 표준 질의어가 없기 때문에 실용성을 고려한 선택으로 추정된다.

2. SPARQL-DL

SPARQL-DL은 OWL-DL을 위한 질의어이다. 표 1은 SPARQL-DL 질의문의 최소 단위인 atom 목록이다. 각각의 atom은 TBox, RBox, ABox atom으로 분류될 수 있다. 표 1의 $C_{(i)}$는 클래스 또는 변수, $p_{(i)}$는 프로퍼티 또는 변수, $a_{(i)}$ 개체(individual) 또는 변수, d는 개체 또는 리터럴 또는 변수이다. 클래스, 프로퍼티, 개체, 리터럴은 고유의 IRI로 표현된다. SPARQL-DL은 Conjunctive 질의 언어로서 SPARQL-DL 질의문은 atom을 접속하여 사용함으로써 구성된다.

SPARQL-DL 질의문은 SELECT 타입 혹은 ASK 타입의 질의문으로 분류할 수 있다. SELECT 타입의 질의문은 변수를 포함한 atom들로 구성된 질의문이며 ASK 타입의 질의문은 변수가 존재하지 않는 atom들로 구성된 질의문이다. 아래의 질의문은 SELECT 타입 질의문의 예이다.

· Type(?x, Student), Type(?x, ?C), SubClassOf(?C, Employee)

위 질의문은 학생이면서 동시에 직원인 학생들 x와 그 조건을 만족하는 학생들의 직책 목록 C를 구하라는 의미이다.

SELECT 타입 질의문을 구성하는 임의의 하나의 atom

내에 존재하는 변수의 결과 집합은 그 atom 내의 타 매개 변수와 그 atom의 의미에 해당하는 OWL 공리에 의해 관계가 설정된 원소들로 구성된다. 복수개의 atom에 동일한 변수가 출현하면 그 변수의 결과 집합은 각각의 atom에서 얻어진 결과 집합들의 교집합 연산에 의한 결과이다. ASK 타입 질의문을 구성하는 각각의 atom의 처리 결과는 true 혹은 false이며 하나의 질의문 내의 모든 atom의 처리 결과가 true이면 질의문 전체의 결과는 true가 된다.

표 1. SPARQL-DL Atom 리스트
Table 1. SPARQL-DL Atom List

TBox Atoms	SubClassOf(C_1,C_2), EquivalentClass(C_1,C_2), DisjointWith(C_1,C_2), ComplementOf(C_1,C_2)
RBox Atoms	SubPropertyOf(p_1,p_2), InverseOf(p_1,p_2), EquivalentProperty(p_1,p_2), ObjectProperty(p), DataProperty(p), Functional(p), Transitive(p), InverseFunctional(p), Symmetric(p)
ABox Atoms	Type(a,C), PropertyValue(a,p,d), SameAs(a_1,a_2), DifferentFrom(a_1,a_2)

3. OWL 온톨로지 생성을 위한 맵핑 규칙

본 논문의 SPARQL-DL 프로세서가 추출하는 OWL 온톨로지는 선행 연구 [16]의 맵핑 규칙에 의해 RDB로부터 생성되는 OWL 온톨로지의 형상을 따른다. 본 절에서는 본론을 이해하는 데 있어서 필요한 내용만을 기술한다. 맵핑 규칙이 적용되기 위한 RDB 스키마의 조건은 '제 1 정규형에 있으며 스키마 내의 모든 테이블에 primary key 제약 조건이 가해져 있는 스키마'이다.

3.1 RDB와 OWL 온톨로지 요소 사이의 맵핑

표 2. RDB와 OWL 온톨로지 요소 간 맵핑
Table 2. mapping between RDB and OWL Ontology

RDB		OWL 온톨로지
스키마	테이블	클래스 cls_T
	key 컬럼	클래스 cls_C, 오브젝트 프로퍼티 op_C, 데이터 프로퍼티 dp_C, XML Schema Datatype(XSD) dt_C
	non-key 컬럼	데이터 프로퍼티 dp_C, XSD dt_C
인스턴스	레코드	개체 ind_T
	key 컬럼 필드	개체 ind_C, 리터럴 lit
	non-key 컬럼 필드	리터럴 lit

표 2는 맵핑 규칙에 의한 RDB와 OWL 온톨로지 두 모델 요소 사이의 맵핑 관계를 보여준다. 표 2에서 사용된 아래 첨자 T는 임의의 테이블을 의미하며 아래 첨자 C는 임의의 컬럼을 의미한다. 표 2에서 사용된 용어 key 컬럼의 의미는 primary key 혹은 foreign key 혹은 1개의 컬럼으로 구성된 unique key 컬럼 제약조건에 있는 컬럼을 뜻한다.

3.2 RDB 인스턴스로부터 ABox 요소 생성하기

표 3은 RDB 인스턴스로부터 생성되는 OWL 온톨로지 ABox 요소의 작명 규칙이다. 개체 ind_T는 테이블 T의 각각의 레코드(row)로부터 생성되는 개체이며 개체 ind_C는 key 컬럼 C의 각각의 필드(cell)로부터 생성되는 개체이다. 표 3에서의 개체를 위한 작명 규칙은 IRI의 fragment 파트이다. 리터럴 lit는 테이블 T의 컬럼 C의 하나의 필드 값으로부터 생성된다. dt_C는 컬럼 C의 SQL 데이터타입에 대응하는 XML 스키마 데이터타입이다. 개체 ind_T와 ind_C를 위한 작명 규칙은 선행 연구 발표 이후 수정된 사항이며 본절 이후의 이해를 위해 반드시 필요한 내용이므로 상세히 기술한다.

표 3. ABox 요소를 위한 작명 규칙
Table 3. Naming Rules for ABox Elements

기호	작명 규칙
ind_T	t = $T^R{}_1$[& k = $PK(T^R, C^{PK})$ & v = $VAL(T, C^{PK})$]$_{\|PK(T)\|}$, 여기에서 $C^{PK} \in PK(T)$ 이다. ※ 컬럼 $PK(T^R, C^{PK})$는 오름차순으로 출현한다.
ind_C	t= $TAB(C^R)$ &c= C^R &v= $VAL(TAB(C), C)$
lit	"$VAL(T, C)$"^^dt_C

표 3에서 사용된 함수 $PK(T)$는 테이블 T의 primary key 제약조건에 있는 컬럼들의 집합, 함수 $TAB(C)$은 컬럼 C가 소속된 테이블의 이름, 함수 $VAL(TAB(C), C)$는 테이블 $TAB(C)$에 소속된 하나의 레코드를 대상으로 그 레코드를 구성하는 필드들 중에서 컬럼 C 필드의 값을 의미한다. 개체 ind_T의 fragment는 세 개의 속성 t, k, v로 구성되어 있다. 하나의 속성은 '속성=값' 패턴을 갖는다. 각각의 속성은 기호 '&'로 구분된다. 속성 t는 레코드가 소속된 테이블 T의 루트 테이블명, k는 루트 테이블의 primary key 컬럼명, v는 컬럼 k의 값을 의미한다. 하나의 primary key는 복수개의 컬럼으로 구성될 수 있으므로 primary key 컬

럼의 개수만큼 속성 k와 v는 반복된다. 개체 ind_C의 fragment는 세 개의 속성 t, c, v로 구성되어 있다. 속성 t는 컬럼 C의 루트 컬럼이 소속된 테이블명, c는 컬럼 C의 루트 컬럼명, v는 컬럼 c의 값을 의미한다. 표 4에서 사용된 C^R은 컬럼 C의 루트 컬럼을 의미한다. 루트 컬럼의 정의는 다음과 같다.

정의 1: $REF(...REF(REF(C)))$처럼 임의의 컬럼 C에 대해서 함수 REF를 연쇄적으로 합성할 때 자신이 참조하는 컬럼이 존재하지 않는 최초의 컬럼이 나타나면 그 컬럼이 컬럼 C의 루트 컬럼이다. 여기에서, 함수 $REF(C)$는 컬럼 C가 참조 무결성 제약조건에 의하여 참조하는 컬럼이다.

루트 컬럼의 정의를 기반으로 동일한 루트 컬럼을 갖는 컬럼들을 패밀리 컬럼이라고 명명한다. 그림 1에서 T1 테이블의 ID 컬럼, T6 테이블의 ID 컬럼, T5 테이블의 ID 컬럼, T4 테이블의 SID 컬럼은 T1 테이블의 ID 컬럼을 루트 컬럼으로 하는 패밀리 컬럼이다. 표 3의 수정된 개체 ind_C를 위한 작명 규칙에 의하면 T1 테이블의 ID 컬럼과 T6 테이블의 ID 컬럼의 필드 값이 2인 두 필드는 모두 fragment가 't=T1&c=ID&v=2'인 개체를 생성한다. 패밀리 컬럼들의 필드 값이 동일할 경우 이들 필드 값들은 논리적으로 동일한 의미를 갖기 때문에 이들로부터 생성되는 개체들은 동일한 IRI를 갖도록 설계하였다.

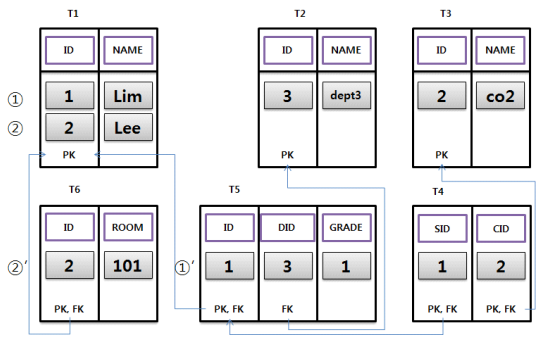


그림 1. 샘플 RDB
Fig. 1. Sample RDB

표 3에서 사용된 T^R은 테이블 T의 루트 테이블을 의미한다. 루트 테이블의 정의하기 이전에 수퍼테이블을 정의하며 이후 루트 테이블을 정의한다.

정의 2: $i \neq j$인 서로 다른 두 테이블 T_i와 T_j가 있다고 가정하자. 두 개의 집합 $PK(T_i)$와 $PK(T_j)$가 $|PK(T_i)| = |PK(T_j)|$ 이고 함수 REF에 대해서 $REF: PK(T_i) \rightarrow PK(T_j)$가 일대일 대응이면, 테이블 T_j는 테이블 T_i의 수퍼테이블이다.

정의 3: $SUPER(...SUPER(SUPER(T)))$처럼 임의의 테이블 T에 대해서 함수 $SUPER$를 연쇄적으로 합성할 때 자신의 수퍼테이블이 존재하지 않는 최초의 테이블이 나타나면 그 테이블이 테이블 T의 루트 테이블이다.

루트 테이블의 정의를 기반으로 동일한 루트 테이블을 갖는 테이블들을 패밀리 테이블이라고 명명한다. 그림 1에서 테이블 T6와 T5의 수퍼테이블은 테이블 T1이다. 테이블 T1은 자신의 수퍼 테이블이 존재하지 않으므로 패밀리 테이블 T1, T5, T6의 루트 테이블이다. 표 3의 수정된 개체 ind_T를 위한 작명 규칙에 의하면 그림 1에서의 레코드 ①과 ①′ 는 모두 fragment가 't=T1&k=ID&v=1'인 개체를 생성한다. 레코드 ②와 ②′ 또한 모두 fragment가 't=T1&k=ID&v=2'인 개체를 생성한다. 그림 1에서 테이블 T4는 컬럼 SID와 CID가 primary key이므로 SID가 1이고 CID가 2인 레코드로부터 fragment가 't=T4&k=CID&v=2&k=SID&v=1'인 개체를 생성한다. 패밀리 테이블 레코드의 primary key 컬럼의 필드 값이 동일하면 이 레코드들은 논리적으로 동일한 대상을 기술하는 것이기 때문에 이들 레코드로부터 생성되는 개체들은 동일한 IRI를 갖도록 설계하였다.

3.3 RDB 스키마로부터 TBox 요소 생성하기

표 4는 RDB 스키마로부터 생성되는 OWL 온톨로지 TBox 요소의 작명 규칙이다. 맵핑 규칙에 의해 생성되는 클래스, 오브젝트 프로퍼티, 데이터 프로퍼티, 개체, 리터럴은 고유의 IRI(Internationalized resource identifier)로 표현된다. 표 3에서 각각의 OWL 요소명은 IRI의 fragment 파트이다.

표 4. TBox 요소를 위한 작명 규칙과 예
Table 4. Naming Rules and Examples for TBox elements

기호	작명 규칙	DB 요소 예	OWL 요소 예
cls_T	T	테이블: sales	sales
cls_C	$TAB(C).C$	테이블: sales 컬럼: id	sales.id
op_C	op_ $TAB(C).C$	테이블: sales 컬럼: id	op_sales.id
dp_C	dp_ $TAB(C).C$	테이블: sales 컬럼: id	dp_sales.id
dt_C	맵핑된 XSD	데이터타입VARCHAR(10)	xsd:string

임의의 key 컬럼 C로부터 생성된 클래스는 참조 무결성 제약조건에 의해 컬럼 C가 참조하는 컬럼으로부터 생성된 클래스의 서브 클래스로 정의되며 루트 컬럼 자격의 컬럼들로부터 생성되는 클래스들은 owl:Thing 클래스의 서브 클래스로 정의된다. 임의의 테이블 T로부터 생성된 클래스는 자신의 수퍼 테이블로부터 생성된 클래스의 서브 클래스로 정의되며 루트 테이블 자격의 테이블들로부터 생성되는 클래스들은 owl:Thing 클래스의 서브 클래스로 정의된다.

3.4 온톨로지 요소간의 연결

그림 2는 RDB 내의 하나의 레코드가 OWL로 맵핑된 형상을 시각화한 것이다. 그림 2에서 타원은 개체, 사각형은 리터럴, 화살표는 프로퍼티를 의미한다. 개체와 개체를 연결한 화살표는 오브젝트 프로퍼티이며 개체와 리터럴을 연결한 화살표는 데이터 프로퍼티이다. 그림 2의 레코드로부터 생성된 개체 ':t=T1&k=ID&v=1'은 그림 1의 레코드 ①과 ①′ 로부터 생성된 것이다. 레코드 ①과 ①′ 는 DB에서 물리적으로 서로 다른 레코드이지만 두 레코드의 primary key 컬럼이 참조 무결성 제약 조건에 있으며 필드 값이 동일하기 때문에 논리적으로 하나의 레코드로 해석한다. 따라서 동일한 IRI가 할당된다. 그러나 이 개체는 두 개의 타입을 갖는다. 첫 번째 타입은 클래스 T1이다. 레코드 ①이 테이블 T1의 레코드이기 때문이다. 두 번째 타입은 클래스 T5이다. 레코드 ①′ 이 테이블 T5의 레코드이기 때문이다. 이처럼 선행 연구의 맵핑 규칙은 패밀리 테이블의 논리적으로 동일한 레코드들 혹은 패밀리 컬럼의 논리적으로 동일한 필드들로부터 생성된 개체에게 동일한 IRI를 할당하지만 그 개체를 생성케 한 레코드의 테이블 혹은 필드의 컬럼 각각으로부터 생성된 클래스 모두를 그 개체의 타입으로 할당한다. 레코드 ①과 ①′ 는 5개의 필드로 구성되어 있기 때문에 각각의 필드 컬럼으로부터 생성된 5개의 프로퍼티 각각은 레코드로부터 생성된 개체를 자신의 컬럼 필드로부터 생성된 개체 혹은 리터럴과 연결된다. 레코드로부터 생성된 개체가 필드로부터 생성된 개체와 연결되었다면 그 필드는 key 컬럼 필드이고 리터럴과 연결되었다면 그 필드는 non-key 컬럼 필드이다. key 컬럼 필드는 개체와 리터럴을 모두 생성하며 이들은 그 key 컬럼으로부터 생성된 데이터 프로퍼티에 의해서 연결된다.

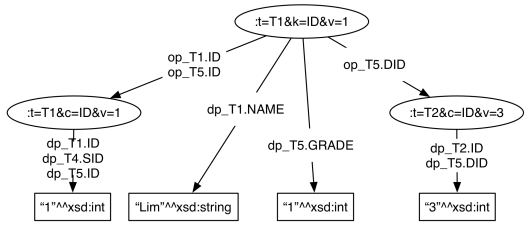


그림 2. ABox 요소 사이의 연결 그래프 I
Fig. 2. Connection Graph between ABox Elements I

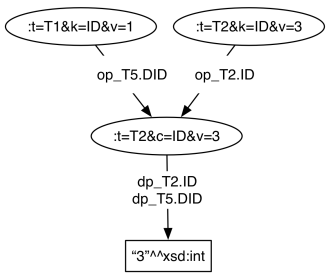


그림 3. ABox 요소 사이의 연결 그래프 II
Fig. 3. Connection Graph between ABox Elements II

그림 3은 패밀리 테이블 관계가 아닌 테이블들의 레코드로부터 생성된 개체들이 동일한 IRI로 표현되는 key 컬럼 필드로부터 생성된 개체에 의해 연결되는 모습을 보여준다. 그림 3에서 개체 't=T2&c=ID&v=3'은 그림 1의 T2 테이블의 ID 컬럼과 T5 테이블의 DID 컬럼의 값이 3인 필드로부터 생성되었다.따라서 이 두 필드를 포함하는 두 개의 레코드로부터 생성된 각각의 개체는 모두 개체 't=T2&c=ID&v=3'과 연결될 수 있다. 그림 3의 형상은 RDB에서 테이블 T2와 T5를 대상으로 한 equi 조인 연산의 결과와 대등하다.

III. 본 론

본 절에서는 본 논문의 SPARQL-DL 프로세서의 설계 및 구현 사항을 기술한다.

1. 시스템 구조 및 동작 흐름

그림 4는 본 논문에서 제안하는 시스템의 구조이다. 시스템을 구성요소의 역할을 시스템 동작 순서에 따라 기술한다. 사용자가 DB 연결을 성공하면 시스템은 RDB의 DB 스키마 전체를 추출하여 RDB Schema Metadata 컴포넌트에 저장한다. 이 과정이 완료되면 시스템은 TBox Generator를 작동시킨다. TBox Generator는 로컬에 저장된 DB 스키마 정보 전체를 입력으로 하며 입력 데이터를 맵핑 규칙에 따라 OWL로 번역한 TBox를 출력으로 한다. 이 과정에서 생성된 맵핑 메타데이터는 Mapping Metadata 컴포넌트에 저장된다. 저장된 맵핑 메타데이터는 질의 처리 과정에서 사용된다. 시스템은 TBox Generator에 의해 생성된 TBox를 DL Reasoner 즉, Tableau 추론기에 적재시킨다. DL Reasoner는 적재한 TBox를 추론한다. 추론기에서 추론이 완료되면 시스템은 사용자의 SPARQL-DL 질의 요청이 있을 때까지 대기한다. 질의문이 요청되면 시스템은 질의문을 Query Metadata 컴포넌트에 전달한다. 이 컴포넌트는 질의문을 분석하여 질의문 자체에 대한 메타데이터를 생성한다. 질의 처리에 참여하는 컴포넌트들은 Query Metadata 컴포넌트로부터 질의문에 대한 정보를 획득한다. 질의문에 대한 분석이 완료되면 Query Scheduler 컴포넌트가 작동한다. 이 컴포넌트는 질의문이 ASK 타입인지 혹은 SELECT 타입인지 여부에 따라 질의문을 구성하는 atom들의처리 순서를 결정하는 알고리즘을 내장하고 있다. 스케줄링 결과에 따라 임의의 하나의 atom이 처리될 시점이 되면 그 atom의 처리는 표 1의 17개의 atom 종류 중 대응하는 Atom Processor에 위임된다. Atom Processor는 처리 대상 atom이 TBox 혹은 RBox atom일 경우 DL Reasoner의 API를 호출하는 방식으로 atom의 처리 결과를 획득하며 처리 대상 atom이 ABox일 경우 RDBMS에 SQL문을 질의하는 방식으로 atom의 처리 결과를 획득한다. SQL Generator 컴포넌트는 ABox atom을 위한 Atom Processor들의 요청에 의해 필요한 SQL 질의문을 생성해주는 역할을 담당한다. 질의문의 타입에 따른 개별 atom의 처리 결과가 무엇이며 이들을 바탕으로 하나의 최종 결과 셋을 완성하는 과정은 다음 절에서 기술한다.

본 논문의 시스템 내부의 모든 컴포넌트들은 Data Source Connector의 컴포넌트들을 통해서 외부 시스템 즉, DL Reasoner 혹은 RDBMS와 통신하도록 설계되어 있다. 현재 본 논문의 시스템은 DL Reasoner와의 통신을 위해서 OWL API [17]을 사용하고 있으며 RDBMS와의 통신을 위해서 JDBC를 사용하고 있다. Data Source Connector의 목적은 OWL API 혹은 JDBC에 변경이 발생하거나 혹은 이들이 대체되더라도 시스템 내부의 컴포넌트이 그 영향으로 인해 수정되지 않도록 하기 위함이다.

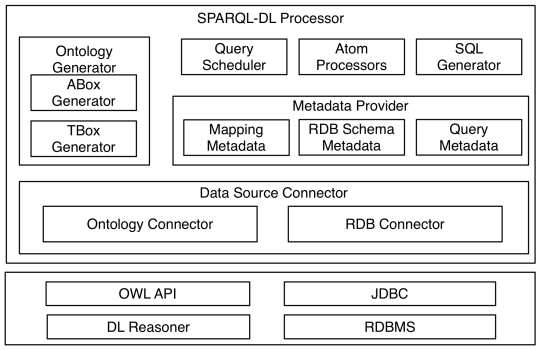


그림 4. 시스템 구조
Fig. 4. System Architecture

2. 질의문 처리 알고리즘

2.1 질의문 처리 알고리즘의 설계 원칙

ABox atom들에 대한 처리는 RDBMS와의 통신을 동반한다. 그러나 RDBMS와의 빈번한 질의응답 과정은 통신상의 오버헤드로 인하여 사용자에게 빠른 응답 시간을 제공할 수 없다. 따라서 본 논문의 시스템은 질의문 타입과 무관하게 하나의 질의문 처리에 대하여 1회의 SQL 질의가 발생하도록 설계하였다.

2.2 ASK 타입 질의문의 처리

그림 5는 ASK 타입 질의문 처리를 위한 알고리즘이며 그림 7은 SELECT 타입 질의문 처리를 위한 알고리즘이다. 그림 5와 7에서 사용된 R0, R1, R2, T0, T1, T2, A0, A1, A2, A3는 모두 atom들의 리스트 이름이다. 이 리스트 변수명의 첫 문자 'R', 'T', 'A'는 각각 그 리스트 내에 있는 atom들이 RBox atom, TBox atom, ABox atom임을 의미하며 뒤이어 나오는 숫자는 그 리스트 내의 atom들이 각각 포함하고 있는 변수의 개수이다. 예를 들어, R0은 변수의 개수가 0개인 RBox atom들의 목록이다. 이 atom들의 리스트는 모두 Query Metadata 컴포넌트로부터 얻어온다.

ASK 타입 질의문은 변수가 없는 atom들로만 구성된 질의문이므로 처리 대상은 R0, T0, A0 리스트이다. 처리 순서는 R0, T0, A0 순으로 진행된다. 그림 5의 06 행에서 호출되고 있는 executeAtoms 함수와 13행에서 호출되고 있는

executAtomsWithoutPresenceCheck() 함수는 인자로 전달된 리스트 내의 각각의 atom들을 하나씩 처리한다. 하나의 atom에 대한 질의 결과는 true 혹은 false이다. 이 함수는 리스트 내의 모든 atom들의 처리 결과가 true일 때 true를 리턴하지만 리스트 내의 임의의 atom에 대한 처리 결과가 false이면 그 이후의 처리되지 않은 atom이 존재할지라도 이들을 처리하지 않고 바로 false를 리턴한다. ASK 타입 질의문은 모든 atom들의 처리 결과가 true일 때 최종 질의 결과가 true가 되기 때문이다.

```
01: Get [R0, T0] from the Query Metadata Component.
02: // [R0, T0] is an array of lists of atoms.
03
04: for var atoms in [R0, T0] {
05:   if (atoms is not empty)
06:     if (executeAtoms(atoms) == false)
07:       return false;
08: }
09:
10: Get A0 from the Query Metadata Component.
11: // A0 is a list of ABox atoms which have no variables.
12: if (A0 is not empty) {
13:   if (executeAtomsWithoutPresenceCheck(A0) == false)
14:     return false;
15:   if (executeAtomsWithPresenceCheck(A0) == false)
16:     return false;
17: }
18:
19: return true;
```

그림 5. ASK 질의문을 위한 스케줄링 알고리즘
Fig. 5. Scheduling Algorithm for ASK queries

그림 1의 DB로부터 생성된 온톨로지를 기반으로 T0 혹은 R0 리스트를 처리하는 과정을 예를 들어 설명하면, ASK 타입 질의문에 DataProperty(:op_T1.ID)라는 atom이 포함되어 있다면 이 atom은 R0 리스트에 소속되어 있을 것이다. 이 atom의 의미는 ":op_T1.ID로 표기되는 IRI가 온톨로지에 데이터 프로퍼티로서 선언되어 있는가?"이다. executeAtoms() 함수에서는 이 atom을 Atom Processor의 변수가 0개인 DataProperty atom을 처리하기 위한 함수로 전달한다. 이 함수에서는 Ontology Connector 컴포넌트의 isDataProperty()라는 함수를 호출하며 인자로 IRI ':op_T1.ID'를 전달한다. 이 함수는 DL Reasoner의 동일 의미의 함수를 호출한다. 이 IRI는 맵핑 규칙에 의하여 오브젝트 프로퍼티로 선언되어 있기 때문에 false를 리턴한다. 결국 executeAtoms() 함수는 false를 리턴하고 이 atom을 포함한 ASK 질의문의 최종 결과는 false이다.

그림 5에서 리스트 A0에 대한 처리는 리스트 R0, T0과는 달리 두 번의 함수 호출로 이루어진다. A0 atom에 대한 처리는 atom 내에 포함된 개체 혹은 리터럴을 생성시킬 수 있는 DB 데이터가 실제로 DB에 존재하는지 여부에 대한 확인 과정이 필요하지만 존재여부 확인 절차 이전에 명백히 false를 리턴하게 되는 조건을 만족하는 atom이 발견된다면 DB와의 통신 과정 없이 최종 결과를 얻을 수 있게 된다. 13행은 A0 리스트 내의 모든 atom들을 순서대로 방문하여 DB에서의 존재 여부 확인 절차 이전임에도 명백히 false를 리턴하는 atom이 존재하는지를 찾는다. 그림 1로부터 생성된 온톨로지에 기초하여 이 경우에 해당하는 예로서 atom Type(:t=T1&c=ID&v=2, :T2)를 들 수 있다. 이 atom의 의미는 첫 번째 인자 개체 IRI가 두 번째 인자 클래스 IRI의 멤버로 온톨로지에 선언되어 있는지를 묻는 것이다. 첫 번째 인자 개체는 T1 테이블의 ID 컬럼의 값이 2인 필드가 존재해야 생성가능하다. 맵핑 규칙에 의하면 이 개체의 타입이 될 수 있는 클래스들은 컬럼 T1.ID의 패밀리 컬럼들로부터 생성된 클래스들 T1.ID, T4.SID, T5.ID, T6.ID이다. 그러나 두 번째 인자는 T2 테이블로부터 생성된 클래스이다. 따라서 첫 번째 인자 개체는 DB에 이 개체와 맵핑된 필드의 존재 여부과 상관없이 T2 클래스의 멤버가 될 수 없다. 즉, 이 atom의 처리 결과는 false이다.

그림 5의 15행은 리스트 A0에 포함된 atom들의 ABox 요소 인자들을 생성케 한 DB 요소가 실제로 존재하는지 여부를 체크하는 과정이다. 이 단계에서는 존재 여부를 확인할 DB 요소가 복수개이더라도 생성되는 SQL 질의문은 1개이다. 그림 6은 이 단계에서 생성하는 SQL 질의문의 예이다. 그림 6 하단은 A0 리스트에 포함된 atom들이며 그림 6의 상단은 하단의 atom들에 포함된 두 개체 IRI의 DB에서의 존재여부를 확인하는 질의문이다. 존재여부를 확인하는 개체가 두 개이기 때문에 SQL문의 WHERE 절에 EXISTS 컨스트럭트가 2번 사용되었다. 즉, 이 단계에서 생성하는 SQL문의

```
SELECT CASE
            WHEN count(*) > 0 THEN 'true'
            ELSE 'false'
       END
FROM DUAL
WHERE EXISTS (SELECT 1
              FROM T4
              WHERE T4.SID = '1') AND
      EXISTS (SELECT 1
              FROM T5
              WHERE T5.DID = '3')
Type(:t=T1&c=id&v=1, :T4.SID), Type(:T2&c=id&v=3, :T5.DID)
```

그림 6. ABox 요소의 존재 확인을 위한 SQL 질의문
Fig. 6. SQL Query for Presence Check of ABox Elements

SELECT와 FROM절은 언제나 동일하며 A0 내의 존재여부를 확인해야 할 ABox 요소의 개수만큼 EXISTS가 사용된다. 이 질의문은 EXISTS 내의 모든 질의문의 결과 행수가 1이상이어야 true이다.

2.2 SELECT 타입 질의문의 처리

SELECT 타입 질의문은 그림 7에서처럼 T1, T2, R1, R2, A1, A2, A3 atom 리스트 순으로 처리한다. 5행~11행의 for 반복 블록에서 변수 atoms는 T1~A3 중에서 하나를 순차적으로 대신한다.

7행에서 호출되는 함수 executeAtom()은 2개의 매개변수를 전달받는다. 첫 번째 매개변수 vertices는 이전 for 반복까지의 atom 리스트들을 처리한 결과물이며 두 번째 매개변수 atoms는 현재 for 반복에서 처리에 편입될 atom 리스트이다. 하나의 for 반복에서 함수 executeAtoms()에 대한 호출이 완료되면 이 함수는 다음 for 반복에서 사용될 새로운 vertices를 생성한다. 그림 8은 executeAtoms() 함수의 실행과정을 보여준다. 기술의 편의를 위하여 그림 8은 이 함수가 그림 7의 for 반복에서 처음으로 호출될 때의 상황이라 가정한다. 즉, vertices는 비어있는 리스트이며 atoms는 5개의 atom을 포함하고 있는 T1 리스트이다. 그림 8에서 하나의 타원은 atom을 의미하며 atom 이름과 변수가 아닌 매개변수는 그림에서 생략하였다. 이 함수가 첫 번째로 수행하는 작업은 입력된 리스트 내의 atom들을 정점으로 하는 변수 이름에 근거한 인접 그래프를 구축하는 것이다. 그림 8 상단의 "처리전" 그래프가 이 작업의 결과에 해당하는 두 개의 컴포넌트로 구성된 인접 그래프의 초기 모습이다. 이 함수는 인접 그래프의 각각의 컴포넌트 내의 임의의 하나의 정점을 선택한 후 그 정점을 시작으로 너비우선탐색방식으로 해당 컴포넌트의 모든 정점을 방문한다. 시작 정점에 방문했을 경우에는 그 정점의 atom을 실행하고 방문을 완료한다. 시작 정점을 제외한 이후 정점 방문에서는 현재 방문한 정점의 atom을 실행한 후 그 실행 결과를 이전 방문에서의 atom 실행 결과와 교집합 연산을 수행한다. 각각의 컴포넌트의 모든 정점에 대한 방문이 완료되면 컴포넌트 당 하나의 결과 집합을 얻게 된다. 즉, executeAtoms() 함수는 그림 8의 상단의 초기 그래프를 그림 8 하단의 "처리후" 그래프처럼 하나의 컴포넌트를 하나의 정점으로 병합시킨 그래프로 변형시킨다. 이 함수는 병합된 각각의 정점을 요소로 하는 vertices 리스트를 반환하며 반환된 vertices 리스트는 다음 for 반복에서 사용된다. vertices 리스트 내의 정점은 실행결과를 가지고 있는 정점이므로 다음 for 반복에서는 정점에 대한 실행 절차 없이 이전 정점 방문에서의 실행 결과와의 교집합 연산만이 수행된다.

```
01: List vertices; // a empty list for instances of the type Vertex.
02:
03: // {T1, T2, R1, R2, A1, A2, A3} is an array of lists of atoms.
04:
05: for var atoms in {T1, T2, R1, R2, A1, A2, A3} {
06:   if (atoms is not empty) {
07:     vertices = executeAtoms(vertices, atoms);
08:     if (vertices is empty)
09:       return an empty result set;
10:   }
11: }
12:
13: Vertex mergedVertex = getCartesianProduct(vertices);
14:
15: ResultSet resultSet = mergedVertex.getResultSet();
16:
17: if (A1, A2 and A3 are empty)
18:   return resultSet;
19: else
20:   return executeSQLQuery(resultSet.getQuery());
```

그림 7. SELECT 질의문을 위한 스케줄링 알고리즘
Fig. 7. Scheduling Algorithm for SELECT queries

그림 9는 그림 7에서 사용된 Vertex와 ResultSet 클래스의 요약 정보이다. 그림 9의 클래스 Vertex는 그림 7의 vertices 리스트의 요소 타입이다. 클래스 Vertex의 atoms 필드는 병합된 atom들의 목록이며 resultSet 필드는 병합된 처리 결과 셋이다.

그림 9에서 클래스 IRIResultSet과 클래스 SQLResultSet은 인터페이스 ResultSet를 구현한다. 즉, 두 클래스의 인스턴스들은 동시에 ResuletSet 타입의 인스턴스들이다.

하나의 IRIResultSet 인스턴스는 하나의 TBox 혹은 RBox 정점의 처리 결과이다. 하나의 TBox 혹은 RBox atom의 처리 결과 셋은 추론기와의 API 수준의 질의 응답에 의해 획득되며 IRIResultSet 인스턴스 내부의 2차원 테이블에 저장된다. 이 테이블의 컬럼 이름들은 변수명들이다. 이 테이블들의 교집합 연산은 함수 getNaturalJoinedResultSet()의 호출에 대응한다. 따라서 IRIResultSet의 이 함수는 내부에서 직접 두 결과 집합 테이블간의 자연 조인 연산을 수행한다.

하나의 SQLResultSet 인스턴스는 하나의 ABox 정점의 처리 결과이다. ABox atom의 처리 결과 셋은 SQL Generator에서 생성된 SQL 질의문이며 SQLResultSet 인스턴스 내부에 저장된다. ABox 정점의 병합은 함수 getNaturalJoinedResultSet()에서 병합 대상 SQL문들을 기반으로 하여 문자열 조작에 의하여 자연 조인된 새로운

SQL문을 생성하는 과정이다.

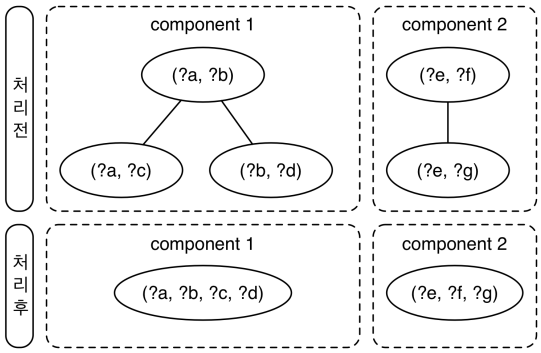


그림 8. 시스템 구조
Fig. 8. System Architecture

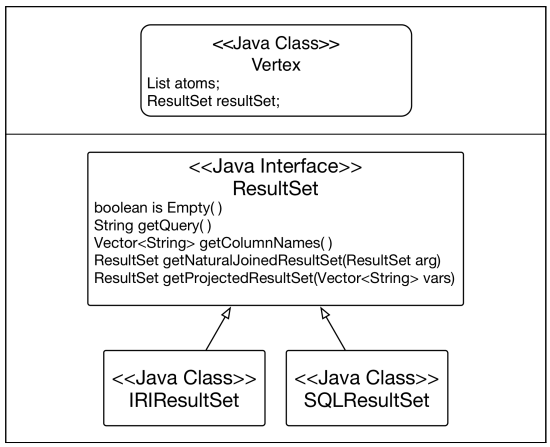


그림 9. Vertex와 ResultSet 클래스 다이어그램
Fig. 9. Class Diagram of Vertex and ResultSet

TBox/RBox 정점과 ABox 정점 사이의 병합은 TBox/RBox 정점의 처리 결과 셋을 SQL 질의문으로 변환한 후 SQL 질의문간의 병합으로 처리한다. TBox/RBox 정점의 처리 결과 셋으로부터 생성된 SQL 질의문은 IRIResultSet 내의 IRI 테이블과 동일한 테이블을 RDBMS에서 질의 처리 동안만 생존하는 임시 view 자격으로 생성시킨다.

그림 7의 13행은 질의문 내의 모든 atom들을 처리하고 이들을 자연 조인에 의해 병합한 결과 그래프의 정점들을 대상으로 cartesian product 연산을 수행한다. 병합 결과 그래프의 정점들은 서로 공유하는 변수가 없기 때문이다. 이 연산이 완료되면 질의문에 대한 하나의 결과 셋을 구하게 된다. 그림 7의 18행에서 리턴하는 최종 결과 셋은 SPARQL-DL 질의문이 ABox atom을 포함하지 않는 경우의 결과 셋이다. 이 경우의 결과 셋은 IRIResultSet 인스턴스 내에 저장된 IRI 테이블이다. 그림 7의 20행에서 리턴하는 최종 결과 셋은 최종적으로 생성된 SQL 질의문을 RDBMS에 질의하여 얻은 결과 셋이다.

2.2 변수를 갖는 ABox Atom을 위한 SQL 질의문

본 논문의 SPARQL-DL 프로세서는 표 1에 포함되어 있는 SPARQL-DL의 모든 atom들에 대하여 나타날 수 있는 모든 매개변수 조합을 처리할 수 있다. 본 절에서는 SELECT 타입 질의문에 포함된 각각의 ABox atom이 처리되었을 때 생성되는 SQL 질의문에 대하여 기술한다. 그림 10~22의 캡션에 사용된 와일드카드 '*'는 모든 변수를 대신한다.

그림 10의 SQL 질의문은 클래스 cls_T의 모든 개체 즉, ind_T들을 구하기 위한 것이다. 이 SQL 질의문은 테이블 T의모든 레코드들을 대상으로 레코드의 primary key 컬럼들의 값들만을 추출한다. 레코드로부터 생성된 개체 IRI는 SELECT 절에서 사용된 함수 recInd()에서 문자열 결합에 의하여 완성된다. 이 함수가 완성하는 개체 IRI는 표 3의 ind_T를 위한 작명 규칙을 준수한다. 이 함수의 인자인 테이블 T의 primary key 컬럼들의 집합은 개체 IRI의 'v' 속성의 값으로서 치환된다.

```
SELECT recInd(PK(T)) FROM T
```

그림 10. Type(*, cls_T)을 위한 SQL 질의문
Fig. 10. SQL Query for Type(*, cls_T)

```
SELECT [DISTINCT] fldInd(C) FROM TAB(C)
[WHERE C IS NOT NULL]
        ※ C∈PK(TAB(C))∧|PK(TAB(C)|=1 이면
                DISTINCT 생략
        ※ C∈NONULL(TAB(C)) 이면 WHERE 절 생략
```

그림 11. Type(*, ds_C)을 위한 SQL 질의문
Fig. 11. SQL Query for Type(*, ds_C)

그림 11의 SQL 질의문은 클래스 cls_C의 모든 개체 즉, ind_C들을 구하기 위한 것이다. 필드 C로부터 생성된 개체 IRI는 SELECT 절에서 사용된 함수 fldInd()에서 문자열 결합에 의하여 완성된다. 이 함수가 완성하는 개체 IRI는 표

3의 ind_C를 위한 작명 규칙을 준수한다. 이 함수의 인자인 컬럼 C는 개체 IRI의 'v' 속성의 값으로서 치환된다. 이 SQL은 중복된 IRI를 생성하지 않도록 컬럼 C의 필드들을 중복되지 않도록 추출한다. 그림 11에서 사용된 함수 $NONULL()$은 인자로 취하는 테이블의 null을 허용하지 않는 컬럼들의 집합을 의미한다.

그림 12의 SQL 질의문은 클래스 'owl:Thing'의 모든 개체 즉, 변환 대상 RDB에서 생성된 모든 개체들을 구하기 위한 것이다. 이 SQL 질의문은 변환 대상 RDB 내의 전체 테이블과 전체 key 컬럼이 아닌 루트 테이블과 루트 컬럼의 자격이 있는 것들만을 대상으로 한다. 그 이유는 표 3을 근거로 모든 개체 IRI의 't', 'k', 'v' 속성의 값은 루트 테이블명, 루트 테이블의 primary key 컬럼명, 루트 컬럼명, 루트 컬럼의 테이블명이기 때문이다. 즉, 중복된 IRI를 추가하지 않기 위해서 루트 테이블들과 루트 컬럼들 만을 대상으로 SQL 질의문을 생성한다. 그림 12의 질의문은 각각의 루트 테이블들과 루트 컬럼들을 기반으로 생성한 질의문들을 UNION 연산자로 결합한 것이다.

Declare Qs. //a set of queries.
for each root table T^R in the RDB Schema
add "**SELECT recInd(**$PK(T^R)$**) FROM** T^R" to Qs.
end for
for each root column C^R in the RDB Schema
add "**SELECT [DISTINCT] fldInd(**C^R**) FROM** $TAB(C^R)$ **[WHERE** C^R **IS NOT NULL]**" to Qs.
end for
Combine Qs with UNION operator.
※ $C^R \in PK(TAB(C^R)) \wedge |PK(TAB(C^R))| = 1$ 이면 DISTINCT 생략
※ $C \in NONULL(T)$이면 WHERE 절 생략

그림 12. Type(*, owl:Thing)을 위한 SQL 질의문
Fig. 12. SQL Query for Type(*, owl:Thing)

Declare Qs. //a set of queries.
add "**SELECT 'owl:Thing' FROM** T **WHERE** $_1[$ C^{PK} **= '**$FLD(ind_T, C^{PK})$**' with 'AND' conjunction** $]_{|PK(T)|}$" to Qs.
for each family table $T^{FM} \in FM(T)$
add "**SELECT '**$cls_{T^{FM}}$**' FROM** T^{FM} **WHERE** $_1[$ $PK(T^{FM}, C^{PK})$ **= '**$FLD(ind_T, C^{PK})$**' with 'AND' conjunction**$]_{|PK(T)|}$" to Qs.
end for
Combine Qs with UNION operator.
※ $C^{PK} \in PK(T)$

그림 13. Type(ind_T, *)을 위한 SQL 질의문
Fig. 13. SQL Query for Type(ind_T, *)

그림 13은 개체 ind_T의 타입들을 구하기 위한 SQL 질의문이다. 개체 ind_T를 생성케 한 레코드가 테이블 T에 존재한다면 기본적으로 ind_T는 'owl:Thing' 클래스의 멤버이며 그 레코드가 테이블 T의 패밀리 테이블들에서도 존재한다면 ind_T 개체는 또한 그 패밀리 테이블들로부터 생성된 클래스들의 멤버이다. 그림 13에서 사용된 함수 $FM()$은 인자가 테이블이라면 패밀리 테이블의 집합을 의미하며 인자가 컬럼이라면 패밀리 컬럼의 집합을 의미한다. 그림 13에서 사용된 함수 $FLD()$는 첫 번째 인자인 개체 IRI의 속성 'v'의 값들 중 두 번째 인자인 primary key 컬럼에 대응하는 값을 반환한다.

그림 14는 개체 ind_C의 타입들을 구하기 위한 SQL 질의문이다. 그림 14는 그림 13에서 레코드가 key 컬럼 필드로 치환되었을 뿐 동일한 원리로 ind_C개체의 클래스들을 얻는다. 그림 14에서 사용된 함수 $FLD()$는 인자인 개체 IRI의 속성 'v'의 값을 반환한다.

Declare Qs. //a set of queries.
add "**SELECT 'owl:Thing' FROM** T **WHERE** C **= '**$FLD(ind_C)$**'**" to Qs.
for each family column $C^{FM} \in FM(C)$
add "**SELECT '**$cls_{C^{FM}}$**' FROM** $TAB(C^{FM})$ **WHERE** C^{FM} **= '**$FLD(ind_C)$**'**" to Qs.
end for
Combine Qs with UNION operator.

그림 14. Type(ind_C, *)을 위한 SQL 질의문
Fig. 14. SQL Query for Type(ind_C, *)

SELECT recInd($PK(TAB(C))$**), fldInd(**C**) FROM** $TAB(C)$ **[WHERE** C **IS NOT NULL]**
※ $C \in NONULL(TAB(C))$이면 WHERE 절 생략

그림 15. PropertyValue(*, op_C, *)을 위한 SQL 질의문
Fig. 15. SQL Query for PropertyValue(*, op_C, *)

그림 15는 오브젝트 프로퍼티 op_C로 연결된 domain 개체와 range 개체의 쌍을 구하는 SQL 질의문이다. 컬럼 C의 필드 값이 null이 아닌 모든 레코드로부터 생성된 개체가 domain 개체이며 그 레코드의 컬럼 C 필드의 값으로부터 생성된 개체가 range 개체이다.

Declare Qs. //a set of queries.
for each family table T^F of the table T
 for each key column C^K in the T^F
  add "**SELECT 'op_{C^K}', fldInd(C^K) FROM** T^F **WHERE** $_1[PK(T^F, C^{PK}) = 'FLD(ind_T, C^{PK})'$ **with 'AND' conjunction**$]_{|PK(T)|}$ **(AND** C^K **IS NOT NULL)**" to Qs.
 end for
 for each non-key column C^{NK} in the T^F
  add "**SELECT '$dp_{C^{NK}}$', lit(**C^{NK}**) FROM** T^F **WHERE** $_1[PK(T^F, C^{PK}) = 'FLD(ind_T, C^{PK})'$ **with 'AND' conjunction**$]_{|PK(T)|}$ **(AND** C^{NK} **IS NOT NULL)**" to Qs.
 end for
end for
Combine Qs with UNION operator.
※ $C^{PK} \in PK(T)$
※ $C^K \in NONULL(T)$이면 WHERE 절 생략
※ $C^{NK} \in NONULL(T)$이면 WHERE 절 생략

그림 16. PropertyValue(ind_T, *, *)을 위한 SQL 질의문
Fig. 16. SQL Query for PropertyValue(ind_T, *, *)

그림 16은 개체 ind_T가 domain 자격의 개체로서 참여하는 모든 연결의 프로퍼티와 range 자격의 개체 혹은 리터럴들의 쌍들을 구하기 위한 SQL 질의문이다. 테이블 T의 모든 패밀리 테이블을 대상으로 개체 ind_T을 생성케 한 레코드들의 모든 컬럼을 추출한다. 추출 컬럼이 key 컬럼이면 그 컬럼으로부터 생성된 오브젝트 프로퍼티 'op_{C^K}'와 그 컬럼의 값으로부터 생성된 개체의 쌍이 구하고자 하는 연결들 중 하나이다. 추출 컬럼이 non-key 컬럼이면 그 컬럼으로부터 생성된 데이터 프로퍼티 '$dp_{C^{NK}}$'와 그 컬럼의 값으로부터 생성된 리터럴의 쌍이 구하고자 하는 연결들 중 하나이다. 그림 16에서 사용된 함수 lit()은 인자로 취한 컬럼을 기반으로 표 3의 lit을 위한 작명 규칙을 준수하는 리터럴 IRI를 생성한다.

그림 17은 개체 ind_C가 range 자격의 개체로서 참여하는 모든 연결의 프로퍼티와 domain 자격의 개체들의 쌍들을 구하기 위한 SQL 질의문이다. 그림 17은 컬럼 C의 모든 패밀리 컬럼들 중에서 ind_C 개체를 생성케 한 필드들을 포함하는 레코드들의 primary key 필드들을 추출한다.

Declare Qs. //a set of queries.
for each family column C^F of the column C
 add "**SELECT 'op_{C^F}', recInd(**$PK(TAB(C^F))$**) FROM** $TAB(C^F)$ **WHERE** C^F **= '**$FLD(ind_C)$**'**" to Qs.
end for
Combine Qs with UNION operator.

그림 17. PropertyValue(*, *, ind_C)을 위한 SQL 질의문
Fig. 17. SQL Query for PropertyValue(*, *, ind_C)

그림 18의 SQL 질의문은 데이터 프로퍼티 dp_C와 맵핑된 컬럼 C가 key 컬럼인가 아닌가에 따라 달라진다. 컬럼 C가 key 컬럼이면 컬럼 C 필드로부터 생성된 개체가 domain 개체이며 컬럼 C 필드로부터 생성된 리터럴이 range 리터럴이다. 컬럼 C가 key 컬럼이 아니면 컬럼 C를 포함하는 테이블 레코드로부터 생성된 개체가 domain 개체이며 컬럼 C 필드로부터 생성된 리터럴이 range 리터럴이다.

if $C \notin KEY(TAB(C))$ then
 SELECT recInd($PK(TAB(C))$**), lit(**C**) FROM** $TAB(C)$ **(WHERE** C **IS NOT NULL)**
end if
if $C \in KEY(TAB(C))$ then
 SELECT (DISTINCT) fldInd(C**), lit(**C**) FROM** $TAB(C)$ **(WHERE** C **IS NOT NULL)**
end if
※ $C \in PK(TAB(C)) \wedge |PK(TAB(C))| = 1$ 이면 DISTINCT 생략
※ $C \in NONULL(TAB(C))$이면 WHERE 절 생략

그림 18. PropertyValue(*, dp_C, *)을 위한 SQL 질의문
Fig. 18. SQL Query for PropertyValue(*, dp_C, *)

Declare Qs. //a set of queries.
for each family column C^F of the column C
 add "**SELECT (distinct) 'dp_{C^F}', lit(**C^F**) FROM** $TAB(C^F)$ **WHERE** C^F **= '**$FLD(ind_C)$**'**" to Qs.
end for
Combine Qs with UNION operator.
※ $C^F \in PK(TAB(C^F)) \wedge |PK(TAB(C^F))| = 1$ 이면 DISTINCT 생략

그림 19. PropertyValue(ind_C, *, *)을 위한 SQL 질의문
Fig. 19. SQL Query for PropertyValue(ind_C, *, *)

그림 19는 컬럼 C의 패밀리 컬럼 필드들 중에서 값이 개체 ind_C를 생성시킨 필드의 값과 동일한 필드가 존재한다면 그 필드의 컬럼으로부터 생성된 데이터 프로퍼티가 결과 집합에 포함될 프로퍼티이며 그 패밀리 컬럼 필드의 값으로부터 생성된 리터럴이 결과 집합에 포함될 range 리터럴이 됨을 의미한다.

그림 20에서 사용된 함수 XSD()는 리터럴 lit의 타입을 반환하고 함수 VALUE()는 리터럴 lit의 값을 반환한다. 그림 20의 SQL 질의문은 우선 RDB 스키마에 정의되어 있는 컬럼들 중에서 해당 컬럼의 SQL 데이터타입과 맵핑된 XSD가 lit의 타입과 같은 컬럼들을 처리 대상으로 선정한 후 선정된 컬럼으로부터 생성된 데이터 프로퍼티를 결과 집합의 프로퍼티로 포함시킨다. 선정된 컬럼이 key 컬럼이면 그 컬럼 필드들로부터 생성된 개체들을 domain 개체들로 포함시키며

선정된 컬럼이 key 컬럼이 아니면 그 컬럼을 포함하는 테이블 레코드들로부터 생성된 개체들을 domain 개체들로 포함시킨다.

```
Declare Qs. //a set of queries.
for each table T in the RDB Schema
 for each column C in the T
  if dt_C == XSD(lit) then
   if C∈KEY(T) then
    add "SELECT (distinct) 'dp_C', fldInd(C)
FROM T WHERE C = 'VALUE(lit)'" to Qs.
   else
    add "SELECT 'dp_C', recInd(PK(T)) FROM T
WHERE C = 'VALUE(lit)'" to Qs.
   end if
 end for
end for
Combine Qs with UNION operator.
      ※ C∈PK(T)∧|PK(T)|=1 이면 DISTINCT 생략
```

그림 20. PropertyValue(*, *, lit)을 위한 SQL 질의문
Fig. 20. SQL Query for PropertyValue(*, *, lit)

그림 21은 컬럼 C의 필드들 중에서 그 값이 개체 ind_{C^F}의 값과 동일한 필드가 존재한다면 그 필드를 포함하는 레코드들로부터 생성된 개체들이 결과 집합의 domain 개체가 됨을 의미한다.

```
SELECT recInd(PK(TAB(C))) FROM TAB(C)
WHERE C = 'FLD(ind_C^F)'
```

그림 21. PropertyValue(*, op_C, ind_{C^F})을 위한 SQL 질의문
Fig. 21. SQL Query for PropertyValue(*, op_C, ind_{C^F})

```
if C⊄KEY(TAB(C)) then
 SELECT recInd(PK(TAB(C))) FROM TAB(C)
WHERE C = 'VALUE(lit)'
end if
if C∈KEY(TAB(C)) then
 SELECT (DISTINCT) fldInd(C) FROM TAB(C)
WHERE C = 'VALUE(lit)'
end if
      ※ C∈PK(TAB(C))∧|PK(TAB(C))|=1 이면
                                  DISTINCT 생략
```

그림 22. PropertyValue(*, dp_C, lit)을 위한 SQL 질의문
Fig. 22. SQL Query for PropertyValue(*, dp_C, lit)

그림 22는 컬럼 C의 필드들 중에서 그 값이 리터럴 lit의 값과 동일한 필드가 존재할 때, 컬럼 C가 key 컬럼이 아니면 그 필드를 포함하는 레코드들로부터 생성된 개체들이 결과 집합의 domain 개체가 되며 컬럼 C가 key 컬럼이면 그 필드로부터 생성된 개체가 결과 집합의 domain 개체가 됨을 의미한다.

그림 23은 테이블 T의 패밀리 테이블들의 non-key 컬럼들 중 컬럼의 SQL 데이터타입과 맵핑된 XSD가 lit의 타입과 같으며 필드 값이 리터럴 lit의 값과 동일한 필드를 포함하는 컬럼이 존재하면 그 컬럼으로부터 생성된 데이터 프로퍼티가 결과 집합에 포함됨을 의미한다.

```
Declare Qs. //a set of queries.
for each family table T^F of the table T
 for each non-key column C^NK in the T^F
  if dt_C == XSD(lit) then
   add "SELECT 'dp_C^NK' FROM T^F WHERE 1[
PK(T^F, C^PK) = 'FLD(ind_T, C^PK)' with 'AND'
conjunction]|PK(T)| AND C^NK = 'VALUE(lit)'" to Qs.
  end if
 end for
end for
Combine Qs with UNION operator.
                                  ※ C^PK∈PK(T)
```

그림 23. PropertyValue(ind_T, *, lit)을 위한 SQL 질의문
Fig. 23. SQL Query for PropertyValue(ind_T, *, lit)

```
Declare Qs. //a set of queries.
for each family column C^F of the column C
 add "SELECT (distinct) 'dp_C^F' FROM TAB(C^F)
WHERE C^F = 'VALUE(lit)'" to Qs.
end for
Combine Qs with UNION operator.
  ※ C^F∈PK(TAB(C^F))∧|PK(TAB(C^F))|=1 이면 DISTINCT
                                          생략
```

그림 24. PropertyValue(ind_C, *, lit)을 위한 SQL 질의문
Fig. 24. SQL Query for PropertyValue(ind_C, *, lit)

그림 24는 컬럼 C의 패밀리 컬럼들 중 필드 값이 리터럴 lit의 값과 동일한 필드를 포함하는 컬럼이 존재하면 그 컬럼으로부터 생성된 데이터 프로퍼티가 결과 집합에 포함됨을 의미한다.

```
Declare Qs. //a set of queries.
for each family column C^F of the column C, where
C^F∈COL(T^F)
 add "SELECT 'op_C^F' FROM TAB(C^F) WHERE 1[
PK(TAB(C^F), C^PK) = 'FLD(ind_T, C^PK)' with 'AND'
conjunction]|PK(T)| (AND C^F = 'FLD(ind_C)')" to Qs.
end for
Combine Qs with UNION operator.
                                  ※ C^PK∈PK(T)
  ※ C^F∈PK(TAB(C^F))이면 WHERE 절의 (AND C^F =
                              'FLD(ind_C)') 생략
```

그림 25. PropertyValue(ind_T, *, ind_C)을 위한 SQL 질의문
Fig. 25. SQL Query for PropertyValue(ind_T, *, ind_C)

그림 25는 개체 ind_T를 생성할 수 있는 테이블 T의 패밀리 테이블들의 레코드들 중에서 개체 ind_C를 생성할 수 있는 컬럼 C의 패밀리 컬럼들의 필드를 포함하는 레코드가 존재한다면 그 컬럼 C의 패밀리 컬럼들로부터 생성된 오브젝트 프로퍼티들이 결과 집합을 구성함을 의미한다.

그림 26에서 key 컬럼 C는 테이블 T의 패밀리 테이블들 중 하나에 포함되어 있는 컬럼이다. 컬럼 C를 포함하는 테이블 레코드들 중에서 개체 ind_T를 생성할 수 있는 레코드들이 존재하면 그 레코드들의 컬럼 C 필드들로부터 생성된 개체들이 결과 집합을 구성한다. 그림 27은 그림 26과 비교하여 컬럼 C가 non-key 컬럼이라는 것과 결과 집합에 리터럴을 포함시키는 것을 제외하고 처리 방식이 동일하다.

SELECT fldInd(C) **FROM** $TAB(C)$ **WHERE** $_1[PK(TAB(C), C^{PK})$ = '$FLD(ind_T, C^{PK})$' **with 'AND' conjunction**$]_{|PK(T)|}$

※ $C^{PK} \in PK(T)$

그림 26. PropertyValue(ind_T, op_C, *)을 위한 SQL 질의문
Fig. 26. SQL Query for PropertyValue(ind_T, op_C, *)

SELECT lit(C) **FROM** $TAB(C)$ **WHERE** $_1[PK(TAB(C), C^{PK})$ = '$FLD(ind_T, C^{PK})$' **with 'AND' conjunction**$]_{|PK(T)|}$

※ $C^{PK} \in PK(T)$

그림 27. PropertyValue(ind_T, dp_C, *)을 위한 SQL 질의문
Fig. 27. SQL Query for PropertyValue(ind_T, dp_C, *)

SELECT [DISTINCT] lit(C) **FROM** $TAB(C)$ **WHERE** C = '$FLD(ind_{C^F})$'

※ $C \in SUK(T) \vee C \in PK(TAB(C)) \wedge |PK(TAB(C))| = 1$ 이면 DISTINCT 생략

그림 28. PropertyValue(ind_{C^F}, dp_C, *)을 위한 SQL 질의문
Fig. 28. SQL Query for PropertyValue(ind_{C^F}, dp_C, *)

그림 28의 atom이 처리 가능하기 위해서는 개체 ind_{C^F}을 생성케 한 필드의 컬럼 C^F가 컬럼 C의 패밀리 컬럼이어야 한다. 컬럼 C의 필드 값이 개체 ind_{C^F}의 속성 'v'의 값과 동일하다면 그 필드 값으로부터 생성된 리터럴이 결과 집합을 구성한다.

Atom Type(*, *)은 온톨로지에 선언된 모든 클래스에 대하여 그 클래스들의 멤버 개체 목록을 결과 집합으로 한다. 이 atom을 위한 SQL은 그림 10~12의 SQL문을 모든 클래스에 대하여 적용한 후 UNION 연산자로 결합한다.

Atom PropertyValue(*, *, *)은 ABox에 선언된 모든 연결의 목록을 결과 집합으로 한다. 이 atom을 위한 SQL은 RDB의 모든 테이블의 모든 레코드를 1회씩 방문하여 결과 집합을 완성한다.

본 논문의 시스템이 내장한 맵핑 규칙에 의하여 생성된 OWL 온톨로지의 개체들은 IRI가 다를 경우 추론기에 의해 논리적으로도 서로 다르게 식별되며 IRI가 동일할 경우 논리적으로도 동일하도록 공리들이 설정되어 있다. 본 논문에서는 SameAs와 DifferentFrom atom을 위한 결과 집합을 이러한 성질에 기반하여 구성한다.

IV. 실 험

이 장에서는 본 논문에서 제안한 시스템의 질의 처리 성능을 타 시스템과의 비교 실험을 통하여 평가하고자 한다.

비교 대상 시스템은 Tableau 추론기인 Pellet 2.3.1 [18]과 Tableau 추론기를 기반으로 동작하는 SPARQL-DL 프로세서인 derivo GmbH SPARQL-DL engine[19]으로 구성된다. 이 시스템은 OWL API와 Tableaux 알고리즘 기반 추론기와 연동하여 사용할 수 있는 유일한 SPARQL-DL 질의 엔진이다. 비교 시스템은 RDB로부터 물리적으로 생성한 즉, ABox와 TBox 모두를 포함하는 OWL 온톨로지를 추론기에 적재한 후 SPARQL-DL 프로세서를 통해 온톨로지 요소 검색을 제공한다.

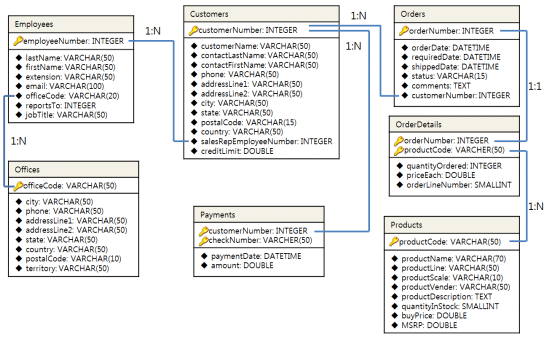


그림 29. 실험 RDB 스키마 다이어그램
Fig. 29. RDB Schema Diagram for Experiment

그림 29는 실험에 사용된 RDB의 스키마이다. 표 5는 그림 29의 스키마를 준수하는 4개의 RDB와 각각의 RDB로부터 생성된 OWL 온톨로지의 정보이다. 표 5의 온톨로지는 본 논문의 시스템의 Ontology Generator 컴포넌트가 물리적으로 생성한 즉, RDB를 OWL 파일로 덤프한 온톨로지이다. 따라서 이 온톨로지는 TBox와 ABox를 포함한다. 추론 전/후의 공리 개수

표 5. 실험 데이터 셋 정보
Table 5. Experiment Data Sets

	레코드 수	DB 크기	온톨로지 크기	추론전 공리수	추론후 공리수
DB(1)	4,297	528 KB	6.25 MB	32,711	37,754
DB(2)	6,929	896 KB	12.4 MB	64,811	74,524
DB(3)	14,274	2.2 MB	18.1 MB	92,469	≒106,000
DB(5)	18,337	3.5 MB	29.5 MB	≒150,000	≒170,000

는 OWL 파일을 Tableau 추론기인 Pellet에 적재하여 추론기로부터 얻어낸 정보이다. DB(3)의 추론 후 공리수와 DB(5)의 추론 전/후 공리수는 추정한 값이다. 그 이유는 Tableaux 추론기는 DB(3) 데이터 셋의 적재에는 성공하였으나 Heap 메모리 부족으로 인하여 추론에는 실패하였으며 DB(5) 부터는 적재 자체에 Heap 메모리 부족으로 인하여 실패했기 때문이다. 이것은 DB(3) 용량부터는 Tableaux 추론기를 통한 온톨로지 서비스가 불가능함을 의미한다.

표 6은 본 실험을 위한 4개의 질의문 목록이다. 질의문의 구성은 SPARQL-DL ABox 질의 처리 성능 평가를 위한 연구 [20]을 따랐다. 질의문 Q1은 TBox와 ABox atom이 혼합되었을 때 질의 엔진의 성능을 평가하기 위함이다. 질의문 Q2는 RBox와 ABox atom이 혼합되었을 때 질의 엔진의 성능을 평가하기 위함이다. 질의문 Q3은 PropertyValue atom에서 술어 위치에 변수가 존재하는 질의이다. 즉, 이 질의는 타 atom에 의해 결정된 주어 ?x와 목적어 ?z의 관계를 찾아야 하는 질의로서 참고로 KAON2의 경우 이러한 종류의 질의를 처리 할 수 없다 [8]. 질의문 Q4는 down-monotoic 변수가 모두 포함된 질의이다. ABox atom인 Type과 PropertyValue의 두 번째 매개변수는 TBox 요소이다. 이 위치의 변수를 down-monotoic 변수라고 한다. 즉, ABox atom에 포함된 TBox 요소이다.

표 6. 실험 질의문 목록
Table 6. Query List for Experiment

번호	질의문
Q1	Type(?x, :payments.customerNumber), Type(?x, ?C), SubClassOf(?C, :customers.customerNumber)
Q2	Type(?z, :offices.officeCode), PropertyValue(:t=employees&k=employeeNumber&v=1323, ?y, ?z), SubPropertyOf(?y, owl:topObjectProperty)
Q3	Type(?x, :customers), PropertyValue(?x, ?y, ?z), Type(?z, :employees.employeeNumber)
Q4	Type(?x, :orders), PropertyValue(?x, ?y, ?w), Type(?w, ?z), SubClassOf(?z, :customers.customerNumber)

실험에 사용된 컴퓨터 시스템은 64비트 Windows 7 서비스 팩 1 운영체제이며 3.0 GHz의 Intel Core i5-3330 CPU와 8 GB의 메인 메모리, 그리고 1 TB의 HDD로 구성된다. 자바 실행 환경은 JRE 7.0이며 Heap 메모리 영역을 기본 1 GB, 최대 2 GB로 설정하였다.

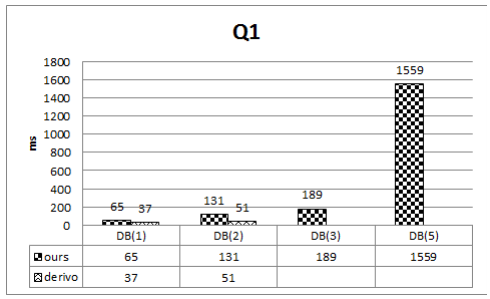


그림 30. 질의문 Q1의 실험 결과
Fig. 30. Experiment Result for Query Q1

그림 30~33은 실험 결과이다. 시간 측정 단위는 1/1000초이다. 격자무늬 막대그래프가 본 논문의 시스템의 처리 결과이고 물결무늬 막대그래프가 비교 시스템의 처리 결과이다. 비교 시스템의 경우 DB(3)과 DB(5)를 대상으로 한 실험이 불가능하였기 때문에 측정값을 표기하지 않았다. 모든 질의문에 대한 실험 결과에서 확인할 수 있듯이 비교 시스템이 서비스 불가능한 DB 용량에 대해서도 본 논문의 시스템은 질의 처리가 가능하였다.

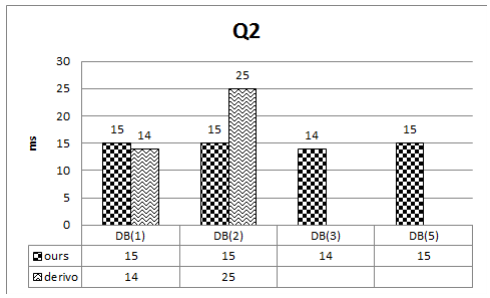


그림 31. 질의문 Q2의 실험 결과
Fig. 31. Experiment Result for Query Q2

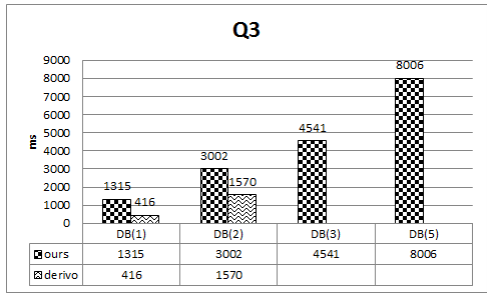


그림 32. 질의문 Q3의 실험 결과
Fig. 32. Experiment Result for Query Q3

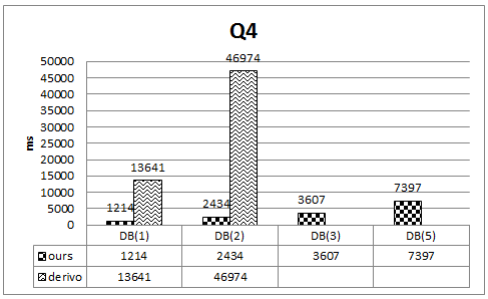


그림 33. 질의문 Q4의 실험 결과
Fig. 33. Experiment Result for Query Q4

비교 시스템이 처리 가능한 DB(1)과 DB(2)를 대상으로 한 실험 결과만을 분석해 보면 모든 온톨로지 요소를 메모리에 적재한 상태로 질의 서비스를 제공하는 비교 시스템의 성능이 DB와의 통신 과정을 포함하는 본 논문의 시스템보다 우수할 것으로 예상되었으나 질의문 Q1과 질의문 Q3만이 비교 시스템이 우수한 성능을 보인 반면 질의문 Q2와 질의문 Q4는 본 논문의 시스템 성능이 우수하였다. 본 논문의 시스템이 보인 질의문 Q2에 대한 실험 결과는 DB 용량이 증가하더라도 응답 시간이 증가하지 않았다. 이것은 Q2 질의문에서 사용된 PropertyValue의 첫 번째 매개변수가 개체이기 때문에 이로 인해 DB에서의 처리 대상 레코드가 개체에 맵핑된 특정 레코드로 한정되기 때문이다. 질의문 Q4의 결과는 Tableau 추론기 기반의 질의 처리가 down-monotonic 변수를 포함한 경우 취약하지만 본 논문의 시스템은 질의문의 이러한 특성에 영향을 받지 않음을 보여준다. 질의문 Q1과 질의문 Q3의 결과를 볼 때 Type과 PropertyValue atom의 매개변수가 모두 변수일 때의 처리에 대한 본 논문 시스템의 개선이 이루어진다면 비교 시스템이 처리 가능한 DB 용량에 대해서도 비교 시스템과 유사한 응답 시간을 제공할 것으로 예상된다.

IV. 결 론

본 논문에서 제안한 시스템은 이미 구축되어 있는 RDB에 OWL을 위한 질의어인 SPARQL-DL을 통해 질의응답을 수행할 수 있도록 한다. 즉, 본 논문의 시스템을 사용하여 사용자들은 RDB에 어떠한 변경을 가하지 않고서도 RDB로부터 OWL 온톨로지 요소를 추출할 수 있게 된다. 이것이 가능한 이유는 제안 시스템이 RDB 스키마에 정의되어 있는 관계형 모델에 기반한 객체들의 관계 및 무결성 제약 조건을 OWL 공리로 번역할 수 있는 기능과 사용자의 SPARQL-DL 질의문을 SQL 질의문으로 번역할 수 있는 기능을 내장하고 있기 때문이다. 본 논문의 시스템과 연동될 수 있는 RDB 스키마의 조건은 '제 1 정규형에 있으며 스키마 내의 모든 테이블에 primary key 제약 조건이 존재하는 스키마'이다.

SPARQL-DL 질의문은 SPARQL-DL 규격에서 정의한 atom들의 접속으로 구성된다. 본 논문에서는 SPARQL-DL 규격에서 정의한 Annotation atom을 제외한 모든 atom이 처리될 수 있도록 구현하였다. RDB 스키마로부터는 부가적인 정보인 annotation을 획득할 수 없기 때문에 구현에서 제외하였다. SPARQL-DL 질의문의 종류는 질의문에 변수를 전혀 포함하지 않는 ASK 타입 질의문과 변수를 포함하는 SELECT 타입 질의문으로 분류할 수 있으며 본 논문에서는 두 종류의 질의문 모두를 처리할 수 있도록 구현하였다. 질의문 종류를 불문하고 RDB와의 연결 수립에 따른 오버헤드를 최소화하기 위하여 본 논문의 SPARQL-DL 프로세서는 입력된 하나의 SPARQL-DL 질의문을 기반으로 단 하나의 SQL 질의문을 생성하도록 구현하였다. 즉, 제안 시스템은 사용자로부터 입력된 하나의 SPARQL-DL 질의문을 처리하는 과정에서 단 한 번의 SQL 질의응답을 발생시킨다.

본 논문에서는 실험을 통하여 제안 시스템이 Tableau 기반 추론기를 기반으로 하여 서비스 할 수 있는 ABox 용량의 한계를 넘는 RDB로부터 생성된 OWL 온톨로지도 서비스가 가능함을 보였다.

마지막으로 향후 연구로는 질의 처리 시 발생하는 메타데이터를 수집하여 이를 이용한 질의 처리 최적화를 수행하는 것이다.

참고로 본 논문의 시스템에 대한 구현물은 다음 URL에 공개되어 있다. https://github.com/jwchoi/OWL4RDB

REFERENCES

[1] B. He, M. Patel, Z. Zhang, and K. C. Chang, "Accessing the deep web," Communication of the ACM, Vol. 50, pp.94-101, May 2007.

[2] C. Bizer, and A. Seaborne, "D2RQ - treating non-RDF databases as virtual RDF graphs," 3rd International Semantic Web Conference, November 2004.

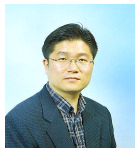
[3] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumuller, "Triplify: lightweight linked data publication from relational databases," Proceedings of the 18th International Conference on World Wide Web, pp. 621-630,

April 2009.
[4] O. Erling, and I. Mikhailov, "RDF Support in the Virtuoso DBMS," Proceedings of the 1st Conference of Social Semantic Web, pp. 59-68, September 2007.
[5] RDB2RDF, http://www.w3.org/2001/sw/rdb2rdf/
[6] I. Horrocks, L. Li, D. Turi, and S. Bechhofer, "The Instance Store: DL Reasoning with Large Numbers of individuals," Proceedings of the Description Logic Workshop, pp. 31-40, June 2004.
[7] C. Chen, V. Haarslev, and J. Wang, "LAS: extending Racer by a large Abox store," Proceedings of the 2005 International Workshop on Description Logics, pp. 200-207, July 2005.
[8] B. Motik, and U. Sattler, "A Comparison of Reasoning Techniques for Querying Large Description Logic Aboxes," Proceedings of LPAR'06, pp. 227-241, November 2006.
[9] E. Sirin, and B. Parsia, "SPARQL-DL: SPARQL query for OWL-DL," Third OWL Experiences and Directions Workshop, June 2007.
[10] M. Li, X. Du, and S. Wang, "Learning Ontology from Relational Database," Proceedings of the 4th International Conference on Machine Learning and Cybernetics. Vol. 6 pp. 3410-3415, August 2005.
[11] Z. Xu, S. Zhang, and Y. Dong, "Mapping between relational database schema and OWL ontology for deep annotation," Proceeding of IEEE/WIC/ACM International Conference on Web Intelligence, pp. 548-552, December 2006.
[12] N. Cullot, R. Ghawi, and K. Yétongno, "DB2OWL: A Tool for Automatic Database-to-Ontology Mapping," In Proceedings of the 15th Italian Symposium on Advanced Database Systems (SEBD 2007), pp. 491-494, June 2007.
[13] F. Cerbah, "Mining the Content of Relational Databases to Learn Ontologies with Deeper Taxonomies," Proceedings of 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology Workshops, pp. 553-557, December 2008.
[14] N. Alalwan, H. Zedan, and F. Siewe, "Generating OWL Ontology for Database Integration," Proceedings of the Third International Conference on Advances in Semantic Processing, pp. 22-31, October 2009.
[15] J. F. Sequeda, and D. P. Miranker, "Ultrawrap: SPARQL execution on relational data," Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 22, pp. 19-39, October 2013.
[16] Ji Woong Choi and Myung Ho Kim, "OWL/Relational Mapping Rules to Use Relational Databases as OWL 2 Web Ontologies," Journal of The Korea Society of Computer and Information, Vol. 16, No. 7, pp. 35-47, July 2011.
[17] The OWL API, http://owlapi.sourceforge.net/
[18] Pellet, http://clarkparsia.com/pellet/
[19] derivo SPARQL-DL engine, http://www.derivo.de/
[20] P. Kremen, and E. Sirin, "SPARQL-DL Implementation Experience," 4th OWL Experiences and Directions Workshop (OWLED-2008 DC), October 2008.

저 자 소 개

최 지 웅

2001: 숭실대학교 컴퓨터학부 학사.
2003: 숭실대학교 컴퓨터학과 석사.
2011: 숭실대학교 컴퓨터학과 박사
현 재: 숭실대학교컴퓨터학부 교수
관심분야: 컴퓨터공학
Email : iamjwchoi@gmail.com



김 명 호

1989: 숭실대학교 컴퓨터학부 학사.
1991: 포항공과대학교 전자계산학과 공학석사.
1995: 포항공과대학교 전자계산학과 공학박사
현 재: 숭실대학교 컴퓨터학부 교수
관심분야: 컴퓨터공학
Email : kmh@ssu.ac.kr