http://dx.doi.org/10.5392/JKCA.2014.14.10.438

관계형 데이터베이스로부터 생성된 OWL 온톨로지를 위한 탐색기

Navigator for OWL Ontologies Generated from Relational Databases

최지웅, 김명호
숭실대학교 컴퓨터학부

Ji Woong Choi(iamjwchoi@gmail.com), Myung Ho Kim(kmh@ssu.ac.kr)

요약

본 논문은 RDB를 OWL 온톨로지로 번역할 수 있으며 번역된 OWL 온톨로지를 사용자가 GUI를 통해 탐색 가능케 하는 시스템을 제안한다. 이러한 목적을 달성하기 위해서 이 시스템은 두 가지 문제를 극복한다. 첫째, 이 시스템은 DB와 온톨로지 사이의 요소간 매핑을 정의하는 새로운 매핑 알고리즘을 내장하고 있다. 이 알고리즘은 기존의 것들과 비교하여 더 다양한 DB 구조로부터 온톨로지를 생성할 수 있다. 둘째, 이 시스템은 DB로부터 생성한 온톨로지의 ABox를 추론기에 적재하지 않고도 추론기로부터 얻을 수 있는 데이터와 동일한 데이터를 사용자에게 제공한다. Tableau 알고리즘 기반의 추론기들은 큰 볼륨의 ABox 추론을 다차시간에 다루지 못하는 문제가 있다. 이것은 DB부터 번역된 큰 볼륨의 ABox를 갖는 온톨로지는 사실상 서비스 될 수 없음을 의미한다. 하지만 이 시스템은 ABox 요소가 요구될 때마다 추론기로부터 얻을 수 있는 데이터와 동일한 데이터를 획득할 수 있는 SQL 질의문을 내부적으로 실행한다.

■ 중심어 : | 시맨틱 웹 | OWL | 관계형 데이터베이스 | RDF | 온톨로지 | 추론 |

Abstract

This paper proposes a system to translate an RDB into an OWL ontology which enables the users to navigate the ontology in GUI. In order to accomplish the goals mentioned previously, the system overcame two difficulties. First, our system defines a new mapping algorithm to map between DB elements and ontology ones. Comparing with existing solutions, our algorithm is able to generate ontologies from more DB structures. Second, our system provides the same data generated by a reasoner to the users. Note that this operation does not load ABox ontology on a reasoner. In addition, Tableau-based reasoners have the tractability problem on a large ABox (e.g., large ABoxes translated from DBs practically cannot be served). To solve this, our system internally runs SQL queries to retrieve the same data as the one from a reasoner every time ABox elements are queried.

■ keyword : | Semantic Web | OWL | Relational Database | RDF | Ontology | Reasoning |

I. 서 론

차세대 웹으로서의 시맨틱 웹의 비전은 표준 데이터 모델인 RDF(Resource Description Framework)와 OWL(Web Ontology Language)을 가지고 웹 스케일의 데이터 수준에서의 데이터 통합을 이루는 것이다. 시맨

접수일자 : 2014년 08월 19일
수정일자 : 2014년 09월 30일
심사완료일 : 2014년 10월 06일
교신저자 : 김명호, e-mail : kmh@ssu.ac.kr

틱 웹이 현재의 웹과 완전한 별개의 것이 아니라 현재의 웹을 기반으로 한 확장의 개념이기 때문에, 두 웹의 연결을 위해서는 현재의 웹에 데이터를 공급하고 있는 다양한 포맷의 데이터 모델들과 시맨틱 웹이 요구하는 RDF 혹은 OWL 모델간 호환성을 확보해야 한다. 그 중에서 동적 웹 콘텐츠의 70% 이상이 관계형 데이터베이스(RDB)로부터 생성되고 있기 때문에 관계형 모델을 RDF 혹은 OWL 모델로 표현하기 위한 연구는 이 분야에서 상당히 시급하며 중요하다[1].

관계형 모델에서 시맨틱 웹 표준 데이터 모델로의 변환은 두 모델의 요소 간 매핑을 정의한 매핑 규칙에 따라서 이루어지며 변환되는 범위와 시점에 따라서 물리적 변환과 가상적 변환 방식이 있다. 물리적 변환은 DB로부터 공개하고자 하는 범위에 해당하는 전체 데이터를 대상으로 ETL(Extrac, Transform, and Load) 과정을 수행한 후 공개되는 방식이다. 결과적으로, 물리적 변환은 사전 변환이며 변환 결과는 원본 DB와는 다른 별도의 저장소에 물리적으로 저장된다. 가상적 변환은 사용자의 SPARQL과 같은 시맨틱 웹 데이터 모델을 위한 질의 언어 포맷의 질의 요청이 발생할 때마다, 이를 SQL 질의로 재작성한 후 그 질의 결과 셋 만을 변환하여 사용자에게 제공하는 방식이다.

RDB의 RDF 그래프로의 변환을 위한 매핑 규칙 및 물리적/가상적 변환에 관한 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].

RDB의 OWL 온톨로지로의 변환을 위한 기존의 매핑 규칙들[6-10]은 제 3 정규형에 있는 DB 스키마를 변환 대상으로 정의하고 있으나 제 3 정규형을 만족하는 스키마임에도 불구하고 변환이 불가능한 것이 존재한다. 본 논문은 이러한 문제를 해소하는 선행 연구[11]의 매핑 규칙을 구현한다. 즉, 본 논문에서는 선행 연구에서 정의한 매핑 규칙에 의거하여 RDB로부터 OWL 온톨로지를 생성하는 생성기를 구현한다.

OWL 온톨로지를 위한 추론기를 통상 DL(Description Logic) 추론기라고 부른다. DL 추론기는 Tableau 알고리즘에 기반하는데 Tableau 알고리즘은 TBox 추론에 최적화 되어 있어서 큰 볼륨의 ABox 추론에 있어서는 다차시간에 다루지 못하는 문제가 있다[12][13]. 이것은 대용량 인스턴스 데이터를 갖는 DB로부터 번역된 큰 볼륨의 ABox를 갖는 온톨로지는 현실적으로 DL 추론기 상에서 서비스 될 수 없음을 의미한다. 본 논문에서는 이러한 문제를 해결하기 위하여 DB 스키마로부터 변환된 TBox 만을 DL 추론기에 적재하며 ABox는 사용자의 탐색에 의해 요구될 때마다 DL 추론기로부터 얻을 수 있는 데이터와 동일한 데이터를 획득할 수 있는 SQL 질의문을 내부적으로 생성, 실행한 DB로부터의 결과 셋을 OWL 포맷으로 번역하여 사용자에게 제공하는 방식을 제안한다. 즉, 본 논문에서 제안하는 시스템은 TBox에 대해서는 물리적 변환 방식을 ABox에 대해서는 가상적 변환 방식을 취한다.

본 논문의 구성은 다음과 같다. 2장에서는 RDB를 OWL 온톨로지로서 서비스하기 위해 필요한 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

1. RDB-to-OWL 매핑 규칙

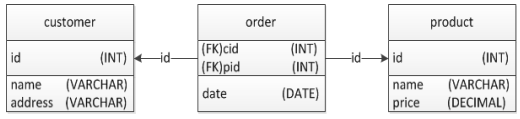


그림 1. junction 테이블을 포함하는 DB 스키마

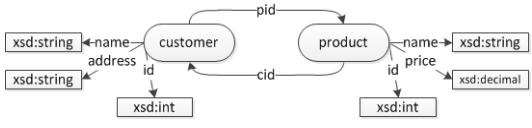


그림 2. 그림 1로부터 유도된 TBox 온톨로지

RDB의 OWL 온톨로지로의 변환을 위한 기존의 매핑 규칙들 [6-10]의 문제점은 관계형 모델의 참조 무결성 제약 조건을 오브젝트 프로퍼티(object property)로 표현하기 때문에 컬럼의 개수가 세 개 이상인 junction 테이블을 변환하지 못한다는 것이다. [그림 1]에서 order 테이블과 같은 테이블을 junction 테이블이라고 한다. 기존의 매핑 규칙들은 date 컬럼이 없는 order 테이블에 대해서는 [그림 2]와 같은 TBox 온톨로지를 생성할 수 있다. [그림 2]에서 타원은 클래스, 사각형은 리터럴의 타입, 화살표는 프로퍼티를 의미한다. [그림 2]의 pid와 cid 객체 속성이 customer와 product 클래스를 연결하고 있음을 볼 수 있는데 이것은 기존의 매핑 규칙들이 junction 테이블로부터는 클래스를 생성하지 않으며 junction 테이블에 연결되어 있는 테이블들로부터 생성한 클래스들을 junction 테이블의 외래키 컬럼으로부터 생성한 오브젝트 프로퍼티들로 직접 연결하는 방식을 취하기 때문이다. 이러한 방식은 order 테이블의 date 컬럼으로부터 생성될 date 데이터 프로퍼티(data property)의 정의역으로 지정될 수 있는 클래스가 존재하지 않게 된다. 따라서 date 컬럼이 변환으로부터 누락된다.

기존의 매핑 규칙들[6][9][10]은 RDB 스키마로부터 TBox 온톨로지를 생성하는 규칙들만 정의하고 있다. 즉, 이 규칙들은 RDB 인스턴스로부터 ABox 온톨로지를 생성하는 규칙은 포함하고 있지 않다. 기존의 매핑 규칙들[7][8]은 RDB 인스턴스로부터 ABox 온톨로지를 생성하는 규칙을 포함하고 있으나 DB의 레코드에서 생성된 개체(individual)간의 식별을 위한 공리가 부재하다. 즉, RDB에서는 기본기 무결성 제약 조건에 의해 레코드간의 식별이 가능하나 OWL 온톨로지에서는 불가능하다는 의미이다. OWL 온톨로지의 하나의 개체는 하나의 IRI로 표현된다. OWL은 열린계 가정 기반이므로 서로 다른 IRI를 갖는 두 개체가 IRI가 다르다는 근거로 무조건 논리적으로 다르다고 판단하지 않으며 서로 다른 IRI의 두 개체가 논리적으로 동일한 개체로 판단 될 수 있다.

2. 대용량 ABox 추론을 지원하는 추론기

RDB에서 유도된 대용량 ABox를 갖는 OWL 온톨로지는 Tableau 알고리즘 기반의 추론기 상에서는 ABox 요소 추출을 위한 질의에 대하여 합리적인 응답 시간을 보장할 수 없다. 이러한 문제를 해결하기 위하여 Instance Store[12]와 LAS[13]와 같은 추론기는 내부적으로 TBox 온톨로지는 기존의 DL 추론기 상에 적재시키고 ABox 온톨로지는 RDBMS에 적재시키는 방식을 사용한다. 이들은 ABox 저장을 위한 고유의 DB 스키마를 갖는다. 따라서 이것은 이것들을 가지고 이미 구축되어 있는 RDB를 OWL 온톨로지로서 서비스하기 위해서는 원본 RDB를 Instance Store 혹은 LAS 고유의 DB 스키마를 준수하는 DB로 변환해야 하므로 반드시 ETL 과정이 요구된다. 즉, 이들 추론기의 사용은 RDB에서 OWL 온톨로지로의 물리적 변환 방식만이 가능하다는 의미이다.

Instance Store와 LAS와는 달리 Tableau 알고리즘의 사용을 완전히 배제한 KAON2[14] 추론기는 입력된 DL 계열 온톨로지를 내부적으로 disjunctive datalog program으로 변환하여 서비스하는 방식을 사용한다. KAON2는 RDB로부터 가상적 변환 방식으로 OWL 온톨로지를 생성할 수 있는 방법을 제공하나 수작업에 의한 RDB 요소와 OWL 요소간의 매핑을 요구하며 두 모델간 매핑에 대한 명세가 공개되어 있지 않기 때문에 변환 가능한 DB 스키마의 범위와 변환된 온톨로지의 표현력 수준에 대하여 논하기 어렵다.

III. 시스템 설계

1. 시스템 요구 사항

본 논문에서 제안하는 시스템의 목적은 RDB로부터 가상적 변환 방식에 의해 OWL 온톨로지를 생성하고 생성된 온톨로지를 사용자로 하여금 GUI 상에서 탐색할 수 있도록 하는 것이다. 이러한 목적을 달성하기 위하여 제안하는 시스템은 다음과 같은 요구 사항을 만족시켜야 한다.

① RDB 스키마로부터 OWL TBox 온톨로지를 물리적으로 생성할 수 있어야 한다.
② RDB 인스턴스로부터 OWL ABox 온톨로지를 가상적으로 생성할 수 있어야 한다.
③ 두 데이터 모델 요소간 매핑에 의한 대응 관계를 제공할 수 있어야 한다.
④ 두 데이터 모델의 개별 요소의 속성을 제공할 수 있어야 한다.

2. 시스템 구조

본 논문의 시스템은 앞서 기술한 요구 사항을 만족시키기 위하여 [그림 3]과 같은 구조를 갖는다. 시스템 구성 요소 각각의 역할은 다음과 같다.

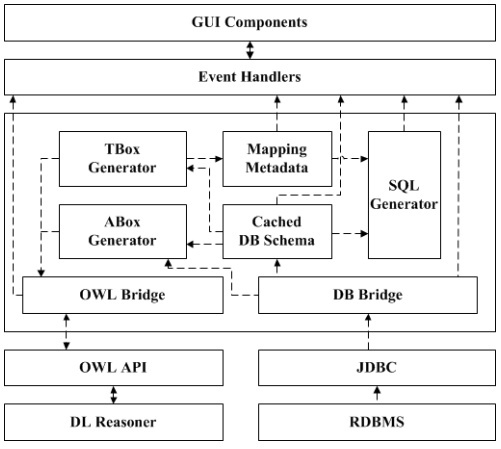


그림 3. 시스템 구조

RDBMS는 OWL 온톨로지 포맷으로 제공하고자 하는 RDB의 스키마와 인스턴스 데이터를 타 모듈에 제공한다. 제안하는 시스템은 JDBC 인터페이스를 사용하여 RDBMS에 접근한다. 본 시스템은 시스템 내의 모듈들이 RDBMS에 접근하고자 할 때, JDBC 메소드를 직접적으로 호출하는 방식 대신 DB Bridge 메소드를 호출하는 방식을 취하도록 설계하였다. 이것은 시스템 내의 모듈들의 JDBC 인터페이스에 대한 독립성을 보장하기 위한 목적이다.

DL Reasoner는 DB 스키마로부터 생성된 TBox 온톨로지를 적재, 추론 후 타 모듈에 제공한다. 제안하는 시스템은 OWL API를 사용하여 추론기에 접근한다. 본 시스템은 시스템 내의 모듈들이 추론기에 접근하고자 할 때, OWL API 메소드를 직접적으로 호출하는 방식 대신 OWL Bridge 메소드를 호출하는 방식을 취하도록 설계하였다. 이것은 시스템 내의 모듈들의 OWL API에 대한 독립성을 보장하기 위한 목적이다.

Cached DB Schema 모듈은 변환 대상 RDB의 스키마 정보를 로컬에 저장한다. 이 모듈의 존재 목적은 RDBMS로의 접근 횟수를 1회로 한정하기 위함이다. 이 모듈은 TBox/ABox 생성기 및 SQL 생성기에서 필요로 하는 DB 스키마 정보를 제공한다.

TBox Generator 모듈은 Cached DB Schema 모듈에서 DB 스키마 정보를 추출하여 TBox 온톨로지를 생성시킨다. 생성된 TBox 온톨로지는 DL Reasoner에 적재되어 추론 과정을 거친 후 사용된다. ABox Generator 모듈은 DB 인스턴스 데이터를 바탕으로 ABox 온톨로지를 물리적으로 생성한다. 가상적 변환 방식을 사용할 경우 이 모듈은 사용되지 않는다.

Mapping Metadata 모듈은 TBox 온톨로지 생성 과정에서 발생하는 두 데이터 모델 요소 간 대응 관계를 저장한다. 이 메타데이터는 GUI 상에서 사용자에 의한 두 데이터 모델 요소 간 대응 관계 조회 이벤트 발생 및 SQL 생성 시 사용된다.

SQL Generator 모듈은 GUI 상에서 사용자에 의한 ABox 온톨로지 요소 조회 이벤트 발생 시 필요한 SQL 질의문을 생성시킨다.

GUI 컴포넌트 상에서 발생하는 이벤트들은 대응하는 이벤트 핸들러에 의해 처리된다. 제안하는 시스템에서 제공하는 이벤트의 범주는 3가지로 분류된다.

① 관계형 데이터 모델 관점에서의 RDB 탐색 이벤트
② OWL 데이터 모델 관점에서의 RDB 탐색 이벤트
③ 두 데이터 모델 요소 간 매핑에 의한 대응 관계 조회 이벤트

범주 ①의 이벤트들은 전형적인 SQL 질의 도구에서 제공하는 기능들에 해당한다. 즉, 테이블 목록 조회, 테이블의 컬럼 목록 조회, 테이블 혹은 컬럼 수준의 무결성 제약 조건 조회, SQL을 사용한 DB 인스턴스 데이터 조회 기능들이다.

범주 ②의 이벤트들은 TBox 요소 탐색과 ABox 요소 탐색 이벤트로 분류된다. TBox 요소는 클래스, 오브젝트 프로퍼티, 데이터 프로퍼티를 의미하며 ABox 요소는 개체를 의미한다. 따라서 TBox 요소 탐색 이벤트는 TBox 요소들의 상속 구조 조회, TBox 요소별 연관된 공리 목록 조회를 위한 이벤트들이며 ABox 요소 탐색 이벤트는 TBox 요소별 연관된 ABox 요소 목록 조회 그리고 ABox 요소별 연관된 ABox 요소 목록 조회를 위한 이벤트들이다.

범주 ③의 이벤트들은 GUI 상에서 선택된 특정 데이터 모델 요소의 다른 데이터 모델 하의 매핑된 요소를 조회해준다. [표 1]은 두 데이터 모델 간 요소 타입에 따른 매핑 관계를 보여준다. [표 1]은 시스템이 내장한 매핑 규칙에 근거한다. key 컬럼은 primary key, foreign key, unique key 제약 조건에 있는 컬럼을 의미하며 non-key 컬럼은 세 가지 key 제약 조건에 있지 않은 컬럼을 의미한다. 본 논문의 매핑 규칙은 unique key 제약 조건에 대해서 1개의 컬럼을 대상으로 한 unique key 제약 조건만 매핑의 대상으로 한다. 그 이유는 복수개의 컬럼을 대상으로 한 unique key 제약 조건은 RDBMS의 구현에 따라 null 값이 포함된 경우 유일성 판단에 차이가 존재하기 때문이다.

표 1. 데이터 모델 사이의 요소 타입별 매핑 관계

RDB 요소		Ontology 요소
테이블		클래스
컬럼	Key 컬럼	클래스, 오브젝트 프로퍼티, 데이터 프로퍼티
	Non-Key 컬럼	데이터 프로퍼티

3. 시스템 동작 시나리오

본 논문의 시스템은 사용자가 DB에 로그온 하는 것으로 시작된다. 연결이 성공하게 되면, 시스템은 DB의 스키마 정보를 Cached DB Schema 모듈로 옮긴다. TBox Generator는 로컬 DB 스키마 정보를 바탕으로 TBox 온톨로지를 생성한 후 이를 DL Reasoner에 적재시키며 DL Reaonser는 이를 추론하여 확장시킨다. 이 과정이 완료되면, 사용자는 GUI 상에서 DB 스키마 정보에 대하여 두 가지 데이터 모델 관점에서 탐색이 가능하다. 사용자는 TBox 요소 탐색 과정 중 특정 TBox 요소와 연관된 ABox 요소들을 조회할 수 있으며 조회 결과 중에서 특정 ABox 요소를 선택하여 그와 연관된 다른 ABox 요소들을 조회할 수 있다.

IV. 시스템 구현

본 절에서는 RDB로부터 OWL 온톨로지를 생성하기 위하여 정의한 선행 연구의 매핑 규칙[11]을 구현하는 알고리즘과 사용자가 요구하는 ABox 요소를 즉시 제공하기 위하여 시스템 내부에서 실행되는 SQL 질의문들의 생성 알고리즘을 분류별로 기술한다.

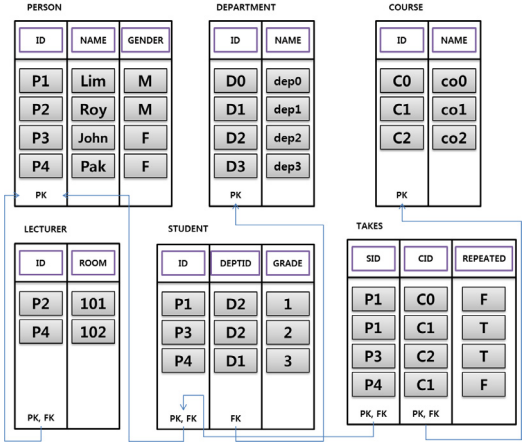


그림 4. college 데이터베이스

[그림 4]의 college 데이터베이스는 본 절과 본 절 이후에서 본 논문의 시스템이 RDB로부터 생성하는 OWL 온톨로지 형상에 관한 구체적인 예를 제시할 목적으로 사용될 변환소스 데이터베이스이다.

1. 매핑 규칙의 구현

본 절은 선행 연구[11]에서 정의한 RDB에서 OWL 온톨로지를 생성하기 위한 매핑 규칙의 구현 내용을 기술한다. [그림 5]는 RDB 스키마로부터 OWL TBox 온톨로지를 생성하기 위한 알고리즘이다. [그림 6]과 [그

림 7]은 [그림 5]의 알고리즘의 일부로써 지면 관계상 분리하였다. [그림 8]은 RDB 인스턴스로부터 OWL ABox 온톨로지를 생성하기 위한 알고리즘이다. TBox/ABox 생성을 위한 알고리즘에서 사용한 OWL 구문은 OWL Functional Syntax[15]이다.

[표 2]는 매핑 규칙을 구현한 온톨로지 생성 알고리즘에서 DB 스키마 정보를 추출하기 위하여 사용되는 함수들의 목록과 그 의미이다.

표 2. 함수의 의미

함수	의미
$SUPER(T)$	테이블 T의 수퍼테이블
$DOM(C)$	컬럼 C의 SQL 데이터 타입
$REF(C)$	컬럼 C가 참조하는 컬럼
$PK(T)$	테이블 T의 primary key 컬럼들의 집합
$FK(T)$	테이블 T의 foreign key 컬럼들의 집합
$SUK(T)$	테이블 T의 single-column unique key 컬럼들의 집합
$KEY(T)$	$PK(T) \cup FK(T) \cup SUK(T)$
$NONULL(T)$	테이블 T의 컬럼들 중 NOT NULL 제약 조건에 있는 컬럼들의 집합

[표 2]의 $SUPER(T)$는 테이블 T의 수퍼테이블을 의미한다. 수퍼테이블은 다음과 같이 정의된다.

정의 1: $i \neq j$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.

[그림 4]에서 테이블 PERSON은 테이블 LECTURER, STUDENT의 수퍼테이블이다. 테이블 LECTURER와 STUDENT의 primary key 컬럼인 ID가 테이블 PERSON의 primary key 컬럼인 ID를 참조하기 때문이다.

TBox를 생성하는 알고리즘은 DB 스키마에 정의되어 있는 모든 테이블 스키마를 순서에 상관없이 한 번씩 방문하여 테이블 수준의 변환을 처리하며 방문한 테이블 스키마의 모든 컬럼을 순서와 무관하게 각각 한 번씩 방문하여 컬럼 수준의 변환을 처리하는 구조이다.

[그림 5] 알고리즘의 시간 복잡도는 RDB 스키마 S 내의 테이블 스키마의 개수를 N_T, 각각의 테이블 스키마 T내의 컬럼 개수의 총합을 N_C라고 할 때 $\Theta(N_T + N_C)$이며 공간 복잡도는 입력 데이터인 RDB 스키마 S를 제외하고 추가적으로 입력 크기 증가에 따라 증가하는 공간을 사용하지 않으므로 $\Theta(1)$이다.

```
Input: an RDB schema S
Output: a TBox ontology TBox

Declare DC. //a set for disjoint classes
for each table schema T in the S
 Generate a class cls_T from the table T.
 print Declaration(Class(:cls_T))
 Declare KOP. // a set for key object properties
 if there is SUPER(T) then
  Generate a class cls_SUPER(T) from the table SUPER(T).
  print SubClassOf(cls_T cls_SUPER(T))
 else
  Add cls_T to DC.
 end if
 for each column C in the T
  Generate a data property dp_C from the column C.
  print Declaration(DataProperty(dp_C))
  Generate an XML Schema datatype dt_C from the SQL data
type DOM(C).
  print DataPropertyRange(dp_C dt_C)
  if C∉ KEY(T) then
   execute the algorithm in 그림 6.
  else
   execute the algorithm in 그림 7.
  end if
 end for // each column
 if |KOP| 〉 0 then
  print HasKey(:cls_T (all object properties in the KOP) ())
 end if
end for // each table schema
if |DC| 〉 0 then
 print DisjointClasses(all classes in the DC)
end if
```

그림 5. TBox 온톨로지 생성을 위한 알고리즘

```
print DataPropertyDomain(dp_C cls_T)
print SubClassOf(cls_T DataAllValuesFrom(dp_C dt_C))
if C∈ NONULL(T) then
 print SubClassOf(cls_T DataExactCardinality(1 dp_C dt_C))
else
 print SubClassOf(cls_T DataMaxCardinality(1 dp_C dt_C))
end if
```

그림 6. Non-Key Column을 위한 알고리즘

```
Generate a class cls_C from the column C.
Generate an object property op_C from the column C.
print Declaration(Class(cls_C))
print Declaration(ObjectProperty(op_C))
print ObjectPropertyDomain(op_C cls_T))
print ObjectPropertyRange(op_C cls_C))
print DataPropertyDomain(dp_C cls_C))
print HasKey(cls_C () (dp_C))
print SubClassOf(cls_C DataAllValuesFrom(dp_C dt_C))
print SubClassOf(cls_C DataExactCardinality(1 dp_C dt_C))
print SubClassOf(cls_T ObjectAllValuesFrom(op_C cls_C))
if C∈NONULL(T) then
 print SubClassOf(cls_T ObjectExactCardinality(1 op_C cls_C))
else
 print SubClassOf(cls_T ObjectMaxCardinality(1 op_C cls_C))
end if
if C∈PK(T) then
 Add op_C to KOP.
end if
if C∈SUK(T) then
 print InverseFunctionalObjectProperty(op_C)
end if
if C∈FK(T) then
 Generate a class cls_REF(C) from the column REF(C).
 print SubClassOf(cls_C cls_REF(C))
 if there is SUPER(T) and C∈PK(T) then
  Generate an object property op_REF(C) from the column REF(C).
  print SubObjectPropertyOf(op_C op_REF(C))
 end if
 Generate a data property dp_REF(C) from the column REF(C).
 print SubObjectPropertyOf(dp_C dp_REF(C))
else
 Add cls_C to DC.
end if
```

그림 7. Key Column을 위한 알고리즘

ABox를 생성하는 알고리즘은 DB 인스턴스에 저장되어 있는 모든 테이블 인스턴스의 모든 레코드를 테이블 인스턴스 단위로 가져올 수 있는 SQL 문을 생성, 실행한 후 각각의 레코드를 하나씩 방문하여 레코드 수준의 변환을 처리하고 방문한 레코드 내의 모든 필드를 한 번씩 방문하여 필드 수준의 변환을 처리하는 구조이다. [그림 8] 알고리즘의 시간 복잡도는 RDB 스키마 S 내의 테이블 스키마의 개수를 N_T, 각각의 테이블 스키마 T의 컬럼 개수의 총합을 N_C, 그리고 각각의 테이블 스키마 T의 인스턴스 테이블의 레코드 개수의 총합을 N_R이라고 할 때 $\Theta(N_T+N_CN_R)$이다. [그림 8]의 알고리즘은 임의의 테이블 스키마 T의 인스턴스 테이블의 레코드를 하나씩 가져와서 처리하는 구조를 취하므로 입력 데이터인 RDB 스키마 S를 제외하고 추가적으로 사용하는 공간의 최대 크기는 모든 레코드 중에서 최대 크기를 갖는 레코드를 저장하기 위한 공간의 크기이다. 그러나 이 공간이 입력 크기가 증가함에 따라 함께 증가하지는 않으므로 이 알고리즘의 공간 복잡도는 $\Theta(1)$이다.

```
Input: an RDB schema S
Output: a ABox ontology ABox

Declare DC. //a set for disjoint classes
for each table schema T in the S
 Fetch a table instance under the T.
 for each record of the table instance
  Generate an individual ind_T from the primary key fields of the record.
  if there is not SUPER(T) then
   print Declaration(NamedIndividual(ind_T))
  end if
  print ClassAssertion(cls_T ind_T)
  for each column C in the T
   Select a field corresponding to the C in the record.
   if the field is not null then
    Generate a literal lit from the field.
    if C∈KEY(T) then
     Generate an individual ind_C from the field.
     if there is not REF(C) then
      print Declaration(NamedIndividual(ind_C))
     end if
     print ClassAssertion(cls_C ind_C)
     print DataPropertyAssertion(dp_C ind_C lit)
     print ObjectPropertyAssertion(op_C ind_T ind_C)
    else
     print DataPropertyAssertion(dp_C ind_T lit)
    end if
   end if
  end for
 end for
end for
```

그림 8. ABox 온톨로지 생성을 위한 알고리즘

표 3. TBox 온톨로지 요소 작명 규칙

기호	작명 패턴	DB 요소 예	OWL 요소 예
cls_T	T	테이블: PERSON	PERSON
cls_C	$TAB(C).C$	테이블: PERSON 컬럼: ID	PERSON.ID
op_C	op_$TAB(C).C$	테이블: PERSON 컬럼: ID	op_PERSON .ID
dp_C	dp_$TAB(C).C$	테이블: PERSON 컬럼: ID	dp_PERSON .ID
dt_C	XML Schema datatype	SQL 데이터 타입 VARCHAR(10)	xsd:string

[표 3]은 [그림 5-8]의 알고리즘에서 기호화된 TBox 온톨로지 요소의 작명 규칙을 설명한다. [표 3]에서 사용된 함수 $TAB(C)$은 컬럼 C가 소속된 테이블의 이름이다.

표 4. ABox 온톨로지 요소 작명 규칙

기호	작명 패턴
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 $PK(T^R, C^{PK})$의 나열 순서는 오름차순이다.
ind_C	:t= $TAB(C^R)$ &c= C^R &v= $VAL(T, C)$
lit	"$VAL(T, C)$" ^^dt_C

[표 4]는 [그림 8]의 알고리즘에서 기호화된 ABox 온톨로지 요소의 작명 규칙을 설명한다. [표 4]에서 사용된 T^R은 테이블 T의 루트 테이블을 의미한다. 루트 테이블은 다음과 같이 정의한다.

정의 2: $SUPER(...SUPER(SUPER(T)))$처럼 임의의 테이블 T에 대해서 함수 $SUPER$를 연쇄적으로 합성할 때 자신의 수퍼테이블이 존재하지 않는 최초의 테이블이 나타나면 그 테이블이 테이블 T의 루트 테이블이다.

[표 4]에서 사용된 C^R은 컬럼 C의 루트 컬럼을 의미한다. 루트 컬럼은 다음과 같이 정의한다.

정의 3: $REF(...REF(REF(C)))$처럼 임의의 컬럼 C에 대해서 함수 REF를 연쇄적으로 합성할 때 자신이 참조하는 컬럼이 존재하지 않는 최초의 컬럼이 나타나면 그 컬럼이 컬럼 C의 루트 컬럼이다.

[표 4]에서 사용된 $PK(T^R, C^{PK})$은 루트 테이블 T^R의 primary key 컬럼들 중에서 테이블 T의 primary key 컬럼 중 하나인 C^{PK}의 패밀리 컬럼을 의미한다. 패밀리 컬럼은 다음과 같이 정의한다.

정의 4: 임의의 컬럼 C에 대해서, 컬럼 C의 루트 컬럼과 동일한 루트 컬럼을 갖는 모든 컬럼들을 컬럼 C의 패밀리 컬럼이라고 한다.

[표 4]에서 사용한 함수 $VAL(T, C)$는 테이블 T의 레코드들 중에서 ind_T, ind_C, lit을 변환생성하기 위하여 방문한 현재 레코드를 대상으로 그 레코드의 필드들 중 컬럼 C에 대응하는 필드의 값을 의미한다.

표 5. ABox 온톨로지 요소 작명 예

번호	기호	예
①	ind_T	:t=TAKES&k=CID&v=C0&k=SID&v=P1
②	ind_T	:t=PERSON&k=ID&v=P4
③	ind_C	:t=PERSON&c=ID&v=P1
④	lit	"Lim" ^^xsd:string

[그림 8]의 ABox를 생성하는 알고리즘은 DB 인스턴스 내의 모든 레코드 그리고 key 컬럼의 필드로부터 개체를 생성하고 모든 컬럼의 필드로부터 리터럴을 생성한다. [표 5]의 예 ①은 [그림 4]의 TAKES 테이블의 primary key 컬럼인 SID, CID의 값이 각각 P1, C0로 식별되는 레코드로부터 생성되는 개체의 축약된 IRI이다. [표 5]의 예 ②는 [그림 4]에서 PERSON, STUDENT, LECTURER 테이블의 primary key 컬럼의 값 P4으로 식별되는 레코드로부터 생성되는 개체의 축약된 IRI이다. 세 테이블은 패밀리 테이블이며 그 중에서 PERSON 테이블이 루트 테이블이다. 패밀리 테이블의 정의는 다음과 같다.

정의 5: 임의의 테이블 T에 대해서, 테이블 T의 루트 테이블과 동일한 루트 테이블을 갖는 모든 테이블을 테이블 T의 패밀리 테이블이라고 한다.

패밀리 테이블의 primaray key 컬럼들의 값이 동일하다면 이것은 동일한 대상의 속성이 여러 테이블에 분산되어 저장된 것이다. 따라서 [표 5]의 예 ②는 세 테이블에 분산되어 있는 세 레코드로부터 동일한 IRI를 생성한다. [표 5]의 예 ③는 [그림 4]에서 key 컬럼들인 PERSON.ID, STUDENT.ID 컬럼의 값 P1을 갖는 필드들로부터 생성되는 개체의 축약된 IRI이다. STUDENT.ID의 루트 컬럼이 PERSON.ID이기 때문에 동일한 값을 갖는 필드로부터 생성된 개체는 동일한 IRI를 갖는다. 패밀리 컬럼의 필드의 값이 동일하다면 이것은 동일한 의미를 갖는 값이 참조 무결성 제약 조건을 통해 여러 테이블에 분산 저장된 것이다. 따라서 이들은 동일한 IRI를 생성한다. [표 5]의 예 ④는 [그림 4]에서 non-key 컬럼인 PERSON.NAME의 값 Lim을 갖는 필드로부터 생성되는 typed 리터럴이다.

[표 6]과 [표 7] 내의 이미지는 본 논문의 시스템으로부터 캡처한 것으로 [그림 4]의 DB로부터 생성된 TBox 요소의 계층 구조와 ABox 요소의 목록이다. [표 6]의 좌측은 모든 테이블과 key 컬럼으로부터 클래스가 생성되었음을 보이며 우측은 모든 key 컬럼으로부터 오브젝트 프로퍼티가 생성되었음을 보인다. [표 7]의 우측은 모든 컬럼으로부터 데이터 프로퍼티가 생성되었음을 보이며 우측은 모든 레코드가 key 필드로부터 개체가 생성되었음을 보인다.

표 6. college 온톨로지 요소 목록 1

클래스 계층구조	오브젝트 프로퍼티 계층구조
owl:Thing :course :course.id :takes.cid :department :department.id :lecturer.deptid :student.deptid :person :lecturer :student :person.id :lecturer.id :student.id :takes.sid :takes	owl:topObjectProperty :op_course.id :op_department.id :op_lecturer.deptid :op_person.id :op_lecturer.id :op_student.id :op_student.deptid :op_takes.cid :op_takes.sid

표 7. college 온톨로지 요소 목록2

데이터 프로퍼티 계층구조	개체 목록
owl:topDataProperty :dp_course.id :dp_takes.cid :dp_course.name :dp_department.id :dp_lecturer.deptid :dp_student.deptid :dp_department.name :dp_lecturer.room :dp_person.gender :dp_person.id :dp_lecturer.id :dp_student.id :dp_takes.sid :dp_person.name :dp_student.grade :dp_takes.repeated	:t=course&k=id&v=C0 :t=course&k=id&v=C1 :t=course&k=id&v=C2 :t=department&k=id&v=D0 :t=department&k=id&v=D1 :t=department&k=id&v=D2 :t=department&k=id&v=D3 :t=person&k=id&v=P1 :t=person&k=id&v=P2 :t=person&k=id&v=P3 :t=person&k=id&v=P4 :t=takes&k=cid&v=C0&k=sid&v=P1 :t=takes&k=cid&v=C1&k=sid&v=P1 :t=takes&k=cid&v=C2&k=sid&v=P3 :t=takes&k=cid&v=C1&k=sid&v=P4 :t=course&c=id&v=C0 :t=course&c=id&v=C1 :t=course&c=id&v=C2 :t=department&c=id&v=D0 :t=department&c=id&v=D1 :t=department&c=id&v=D2 :t=department&c=id&v=D3 :t=person&c=id&v=P1 :t=person&c=id&v=P2 :t=person&c=id&v=P3 :t=person&c=id&v=P4

본 논문에서는 선행 연구와는 다른 방식으로 OWL 개체에 IRI를 할당한다. 선행 연구에서는 서로 다른 IRI로 표현되어 있지만 논리적으로는 동일한 개체들이 생성될 수 있었다. 이러한 개체들은 패밀리 테이블들의 primary key 컬럼들의 값이 같은 레코드들 혹은 패밀리 컬럼들의 값이 같은 필드들로부터 생성된다. 본 논문에서는 이러한 개체들이 자신의 루트 테이블 혹은 루트 컬럼으로부터 생성된 개체의 IRI와 동일한 IRI를 갖도록 수정하였다. 이러한 변경의 의도는 이후 절에서 기술할 SQL Generator가 생성하는 SQL문을 간결하게 함과 동시에 질의 결과 셋의 레코드 개수를 줄임으로써 사용자에게 빠른 응답을 제공하기 위함이다.

2. SQL Generator의 구현

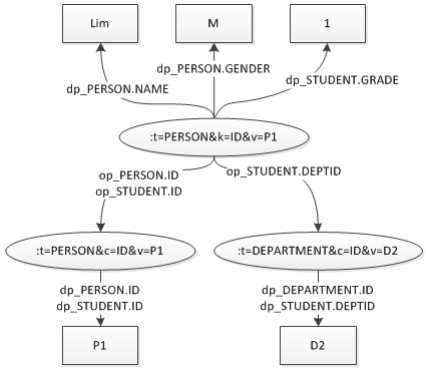


그림 9. 변환된 온톨로지 예

[그림 9]는 [그림 4]의 PERSON 테이블의 primary key 컬럼인 ID의 값 P1으로 식별되는 레코드가 OWL로 변환된 상태를 RDF 그래프와 유사한 형식으로 나타낸 것이다. [그림 9]를 통해서 SQL Generator가 생성하는 SQL 질의문의 의미 이해를 돕고자 한다. [그림 9]에서 타원은 개체, 네모는 리터럴, 둥근 사각형은 클래스, 연결선은 프로퍼티이다. OWL에서는 개체와 개체 사이의 연결선을 오브젝트 프로퍼티라고 하며 개체와 리터럴 사이의 연결선을 데이터 프로퍼티라고 한다. 레코드에서 생성된 개체는 그 레코드의 필드에서 생성된 개체 혹은 리터럴과 연결된다. 연결에 사용된 프로퍼티의 이름을 통해 어느 컬럼으로부터 생성된 개체 혹은 리터럴인지 식별할 수 있다. 레코드로부터 생성된 개체가 개체와 연결되었다면 연결된 그 개체는 그 레코드에서 key 컬럼의 필드에서 생성된 것이며 레코드로부터 생성된 개체가 리터럴과 연결되었다면 연결된 그 리터럴은 그 레코드에서 non-key 컬럼의 필드에서 생성된 것이다. [그림 9]에서 하나의 프로퍼티에 프로퍼티 이름이 두 개 사용된 것은 동일한 IRI를 생성하는 두 개의 패밀리 테이블의 레코드 혹은 두 개의 패밀리 컬럼의 필드가 존재하기 때문이다.

SQL Generator가 생성하는 질의문들은 [그림 8]의 ABox 생성 알고리즘에 의해 생성된 가상의 ABox 온톨로지를 대상으로 [표 8]의 세 가지 타입의 주장(assertion)들 중에서 특정 조건을 만족하는 주장들을 검색하게 해준다. 특정 조건이란 세 가지 타입의 주장 중 검색하고자 하는 주장 타입과 그 주장 타입의 매개변수 중 하나가 결정되어 있는 상태를 의미한다. 검색 결과는 결정되지 않은 나머지 인자의 값들이다. 예를 들어, ClassAssertion의 구체적인 클래스명 cls가 주어지면 SQL Generator는 그 클래스의 모든 개체를 검색할 수 있는 질의문을 생성한다. 검색 대상을 '*'로 표기하면 하나의 검색은 ClassAssertion(cls *)가 된다. [표 8]의 $srcInd$는 op 혹은 dp의 정의역 개체를 의미하며 $targetInd$는 op의 치역 개체, $targetLit$는 dp의 치역 리터럴을 의미한다.

표 8. ABox 온톨로지 주장(assertion) 형식

주장(Assertion) 형식
ClassAssertion($cls\ \ ind$)
ObjectPropertyAssertion($op\ \ srcInd\ \ targetInd$)
DataPropertyAssertion($dp\ \ srcInd\ \ targetLit$)

[표 9]는 본 논문의 시스템에서 SQL 실행에 의해 제공하는 ABox 요소 검색 패턴이다. 본 논문의 시스템은 테이블과 key 컬럼으로부터 클래스를 생성하기 때문에 주어진 클래스의 모든 개체를 검색하는 패턴의 질의문이 그 클래스의 출처의 종류에 따라 다르다. 본 논문의 시스템은 레코드와 key 컬럼 필드로부터 개체를 생성하므로 주어진 개체의 모든 타입 즉, 클래스를 검색하기 위한 질의문 또한 그 생성 출처의 종류에 따라 다르다. 그리고 레코드에서 생성된 개체는 오브젝트 프로퍼티의 정의역 자격으로만 사용되며 key 컬럼 필드로부터 생성된 개체는 오브젝트 프로퍼티의 치역 자격으로만 사용된다.

표 9. ABox 요소 검색 패턴

주장(Assertion) 형식
ClassAssertion(cls_T *), ClassAssertion(cls_C *), ClassAssertion(owl:Thing *)
ClassAssertion(* ind_T), ClassAssertion(* ind_C)
ObjectPropertyAssertion(op_C * *)
ObjectPropertyAssertion(* ind_T *)
ObjectPropertyAssertion(* * ind_C)
DataPropertyAssertion(dp_C * *)
DataPropertyAssertion(* ind_T *), DataPropertyAssertion(* ind_C *)
DataPropertyAssertion(* * lit)

표 10. 개체 및 리터럴 생성을 위한 문자열 결합

recInd($PK(T)$)	concat(':t=T^R' $_1$[, '&k=$PK(T^R, C^{PK})$&v=', $C^{PK}]_{|PK(T)|}$) ,where $C^{PK} \in PK(T)$
fldInd(C)	concat(':t=$TAB(C^R)$&c=C^R&v=', C)
lit(C)	concat('"', C, '"', '^^', 'dt_c')

[그림 10-21]은 [표 9]에서 나열한 검색 패턴 각각을 위한 SQL 질의문을 설명한다. 각각의 검색 패턴을 위한 질의문의 SELECT 절에 올 수 있는 요소는 cls_T, cls_C, op_C, dp_C, ind_T, ind_C, lit이다. 그 중에서 ABox 요소들 즉, ind_T, ind_C, lit은 SELECT 절에서 문자열 결합 연산에 의해 완성된다. [표 10]은 이들 요소의 문자열 결합 연산 패턴을 설명한다. 함수 recInd는 레코드로부터 생성된 개체 문자열을 얻는 함수로서 임의의 테이블 T의 primary key 컬럼 리스트를 인자로 한다. 함수 fldInd는 key 컬럼 필드로부터 생성된 개체 문자열을 얻는 함수로서 임의의 key 컬럼 C를 인자로 한다. 함수 lit는 필드로부터 생성된 리터럴을 얻는 함수로서 임의의 컬럼 C를 인자로 한다. [표 10]의 좌측 세 함수에 대한 [그림 10-21]의 SELECT 절에서의 호출은 실제 구현에서는 [표 10] 우측의 concat 함수 호출문으로 치환되어있다.

[그림 10]은 테이블 T의 모든 레코드의 primary key 컬럼의 필드들을 추출한다.

SELECT recInd($PK(T)$) FROM T

그림 10. ClassAssertion(cls_T *) 질의문

[그림 11]은 컬럼 C의 필드들을 중복되지 않도록 추출한다. 중복된 필드는 중복된 IRI를 생성하기 때문이다.

SELECT [DISTINCT] fldInd(C) FROM $TAB(C)$ [WHERE C IS NOT NULL]
※ $C \in PK(TAB(C)) \wedge |PK(TAB(C)| = 1$ 이면 DISTINCT 생략
※ $C \in NONULL(TAB(C))$ 이면 WHERE 절 생략

그림 11. ClassAssertion(cls_C *) 질의문

[그림 12]는 DB 내의 루트 테이블들과 루트 컬럼들만을 대상으로 한다. 그 이유는 질의 결과 셋에 개체에 대한 중복된 IRI를 추가하지 않기 위해서이다. 최종 질의문은 루트 테이블을 기반으로 생성한 질의문들과 루트 key 컬럼을 기반으로 생성한 질의문들을 UNION 연산자로 결합한다.

Declare Qs. //a set of queries.
for each root table T^R in the S
add "SELECT recInd($PK(T^R)$) FROM T^R" to Qs.
end for
for each root column C^R in the S
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. ClassAssertion(owl:Thing *) 질의문

[그림 13]은 ind_T 개체의 원천 레코드가 존재한다면 ind_T 개체는 기본적으로 owl:Thing 클래스의 멤버가 되며 테이블 T의 패밀리 테이블에 동일한 primary key 필드로 식별되는 레코드가 존재한다면 ind_T 개체는 그 패밀리 테이블로부터 생성된 클래스의 멤버가 된다.

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. ClassAssertion(* ind_T) 질의문

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. ClassAssertion(* ind_C) 질의문

[그림 14]는 [그림 13]에서 레코드가 key 컬럼 필드로 치환되었을 뿐 동일한 원리로 ind_C개체의 클래스들을 얻는다. [그림 13]에서 사용된 함수 FLD는 개체 ind_T의 IRI로부터 컬럼 C^{PK}의 값을 반환하며 [그림 14]에서 사용된 함수 FLD는 개체 ind_C의 IRI로부터 필드 값을 반환한다.

[그림 15]는 $TAB(C)$ 테이블의 컬럼 C의 필드가 null이 아닌 모든 레코드로부터 레코드 개체의 IRI를 완성하기 위하여 primary key 컬럼들의 필드 값과 key 컬럼 개체의 IRI를 완성하기 위하여 컬럼 C의 필드 값을 추출한다.

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

그림 15. ObjectPropertyAssertion(op_C * *) 질의문

[그림 16]은 테이블 T의 모든 패밀리 테이블을 대상으로 ind_T 개체의 원천 레코드와 동일한 primary key 필드를 갖는 레코드들 중에서 null이 아닌 모든 key 컬럼 필드를 추출한다.

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}' , fldlnd(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
end for
Combine Qs with UNION operator.
※ $C^{PK} \in PK(T)$
※ $C^K \in NONULL(T)$ 이면 WHERE 절 생략

그림 16. ObjectPropertyAssertion(* ind_T *) 질의문

[그림 17]은 컬럼 C의 모든 패밀리 컬럼을 대상으로 ind_C 개체의 원천 필드와 동일한 필드들과 그 필드들을 포함하는 레코드들의 primary key 필드들을 추출한다.

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

그림 17. ObjectPropertyAssertion(* * ind_C) 질의문

if $C \notin KEY(TAB(C))$ then
SELECT reclnd($PK(TAB(C))$), lit(C) FROM $TAB(C)$ [WHERE C IS NOT NULL]
end if
if $C \in KEY(TAB(C))$ then
SELECT [DISTINCT] fldlnd(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. DataPropertyAssertion(dp_C * *) 질의문

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

그림 19. DataPropertyAssertion(* ind_T *) 질의문

[그림 18]은 컬럼 C의 key 컬럼 여부에 따라 서로 다른 질의문을 생성한다. 컬럼 C가 key 컬럼이면 컬럼 C의 필드만을 추출하나 컬럼 C가 key 컬럼이 아니면 $TAB(C)$ 테이블의 primary key 컬럼들의 필드를 추출한다.

[그림 19]의 질의문은 [그림 16]의 질의문과 동일한 생성 원리를 갖는다. 차이점은 패밀리 테이블을 대상으로 key 컬럼이 아닌 non-key 컬럼의 필드들을 추출하는 것이다.

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

그림 20. DataPropertyAssertion(* ind_C *) 질의문

Declare Qs. //a set of queries.
for each table T in the S
for each column C in the T
if dt_C == XSD(lit) then
if $C \in KEY(T)$ 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.
※ $C \in PK(T) \wedge |PK(T)| = 1$ 이면 DISTINCT 생략

그림 21. DataPropertyAssertion(* * lit) 질의문

[그림 20]은 ind_C의 필드 값과 동일한 값을 갖는 패밀리 컬럼 필드를 중복 없이 추출한다.

[그림 21]에서 사용된 XSD 함수는 typed 리터럴의 타입을 반환하고 VALUE 함수는 typed 리터럴의 값을 반환한다. [그림 21]의 질의문은 lit의 타입과 같은 dt_C를 생성하는 DB 내의 모든 컬럼들 중에서 lit의 값과 같은 필드가 존재하는 컬럼들에 대하여, 그 컬럼이 key 컬럼이면 그 컬럼 필드만을 추출하고 그 컬럼이 key 컬럼이 아니면 그 필드를 포함하는 레코드의 primary key 필드들을 추출한다.

[그림 22]는 구현된 시스템의 UI이며 Java 언어로 구현되었다. [그림 22]는 [그림 4]의 college 데이터베이스를 OWL 온톨로지로 번역한 모습이다. 영역 ①은 'Database' 탭과 'Ontology' 탭으로 구성되어 있다. 'Database' 탭은 DB 스키마 요소의 계층구조 즉, 테이블의 리스트와 각각의 테이블의 컬럼 리스트를 보여주며 'Ontology' 탭은 TBox 요소 즉, 클래스, 오브젝트 프로퍼티, 데이터 프로퍼티의 계층구조를 보여준다. [그림 22]에서는 현재 'Ontology'탭이 선택되어 있으며 [표 6]과 [표 7]의 내용과 동일하다. 영역 ③은 영역 ①에서 DB 스키마 요소 혹은 TBox 요소 중에서 특정 요소가 선택되었을 때, 그 요소의 속성을 기술해준다. [그림 22]에서의 영역 ③은 특정 OWL 클래스가 선택되었을 때, 그 클래스가 포함된 TBox 공리들을 보여주고 있다. 영역 ②는 ABox 요소의 탐색이 이루어지는 곳이다. 영역 ②의 좌측은 영역 ①에서 선택된 특정 TBox 요소와 연관된 ABox 요소의 리스트를 보여준다. 영역 ②의 우측은 영역 ②의 좌측에서 선택된 특정 개체 혹은 리터럴이 포함된 세 가지 주장들 즉, ClassAssertion, ObjectPropertyAssertion, DataPropertyAssertion들의 목록을 보여준다.

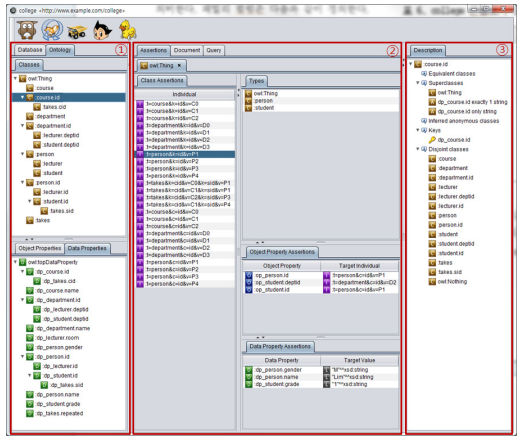


그림 22. 시스템 UI

V. 평 가

1. 매핑 규칙의 평가

RDB의 OWL 온톨로지로의 변환을 위한 기존의 매핑 규칙들[6-10]의 문제점은 관계형 모델의 참조 무결성 제약 조건을 오브젝트 프로퍼티로 표현하기 때문에 컬럼의 개수가 세 개 이상인 junction 테이블을 변환하

지 못한다는 것이었다. 본 논문의 매핑 규칙은 이러한 문제를 해결하기 위하여 key 컬럼으로부터 클래스를 생성하고 key 컬럼간에 참조 무결성 제약조건이 존재할 경우 이들로부터 생성된 클래스들을 SucClassOf 공리에 의해 부모-자식 관계를 맺어줌으로써 동일한 IRI의 개체를 이들 클래스들이 공유할 수 있도록 함으로써 기존 매핑 규칙들의 문제점을 극복하였다. [그림 23]은 [그림 1]의 DB 스키마를 본 논문의 매핑 규칙에 의해 생성한 TBox 온톨로지이다. junction 테이블을 포함하는 DB 스키마의 모든 요소를 누락 없이 온전히 변환하였음을 확인할 수 있다.

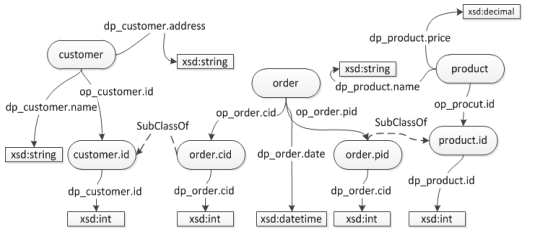


그림 23. [그림 1]로부터 생성된 TBox 온톨로지

2. 대용량 ABox 탐색에 대한 평가

이 절에서는 Tableau 알고리즘 기반의 추론기들이 DB로부터 생성된 대용량 ABox 처리에 있어서 부적합함을 보임과 동시에 본 논문의 시스템은 ABox 용량과 무관하게 서비스 가능함을 실험을 통해 보이고자 한다.

실험은 [표 11]의 구조를 갖는 DB의 인스턴스 용량을 증가시켜가며 이들 DB로부터 본 논문의 매핑 알고리즘에 의해 생성된 온톨로지를 Tableau 알고리즘 기반이며 Java로 구현된 추론기인 Pellet [16]과 본 논문의 시스템에 각각 로드시킬 때의 로드 및 초기 추론 시간을 측정한다. 실험을 위한 최소 크기의 DB는 [표 11]의 product 테이블의 레코드수가 12, customer 테이블의 레코드수가 11, 고객 당 주문은 3건, 주문 당 아이템은 5개로 이루어져있다. DB 크기를 증가시키는 방법은 고객 당 주문과 주문 당 아이템은 고정시키고 product 테이블의 레코드는 5개씩 customer 테이블은 10개씩 증가시켰다. 최소 크기의 DB를 DB(1)이라고 명명하고 괄호 안의 1보다 큰 수는 DB(1)을 증가시킨 횟수이다.

표 11. Shop DB 스키마

category (id, name, parent_id) FK (parent_id) refs category.id
product (ean_code, name, category_id, price, manufacturer, notes, description, image) FK (category_id) refs category.id
order_item(id, order_id, number_of_items, product_ean_code, total_price) FK (order_id) refs order.id FK (product_ean_code) refs (product.ean_code)
order(id, customer_id, total_price, created_at) FK customer_id refs (customer.id)
customer(id, category, salutation, first_name, last_name, birth_date) FK id refs (user.id)
user(id, name, email, password, role_id, active) FK role_id refs (role.name)
role(name)

실험 대상인 두 시스템의 온톨로지 서비스를 위한 구조적 차이로 인하여 Pellet에 로드되는 온톨로지는 물리적 생성 방법에 의한 것이다. 즉, TBox와 ABox를 모두 포함하며 TBox는 [그림 5] 알고리즘의 실행 산출물이며 ABox는 [그림 8] 알고리즘의 실행 산출물이다. 반면 본 논문의 시스템은 초기에 TBox 만을 로드한다.

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. Pellet의 버전은 2.3.1이다.

[표 12][표 13]의 OWL 문서크기는 DB로부터 물리적으로 생성된 OWL 문서의 크기이다. 추론 전 공리수는 Tableau 추론기 상에서 추론 연산이 수행되기 이전의 각각의 OWL 문서가 포함하는 공리의 개수이다. 추론 후 공리수는 Tableau 추론기 상에서 추론 연산이 완료된 이후의 공리의 개수이다.

[표 12]는 Pellet에 DB 기반 온톨로지를 로드시켰을 때의 실험 결과이다. Pellet은 DB(7)부터 메모리 부족 발생으로 인한 추론 중단 현상을 보였다. 즉, DB(7)부터는 추론이 완료되지 못하였기 때문에 DB(7) 이상의 크기를 갖는 DB로부터 생성된 OWL 온톨로지는 서비스 불가능하다.

표 12. Pellet 시스템 실험 결과

DB	OWL문서크기 (KB)	추론전 공리수	추론후 공리수	로딩및추론 시간(ms)
DB(1)	395	2809	3482	5397
DB(3)	1046	7035	8508	10552
DB(5)	1695	11251	13524	19472
DB(6)	2021	13359	16032	26934
DB(7)	2348	15475	-	-
DB(8)	2671	17583	-	-

표 13. 제안하는 시스템 실험 결과

DB	OWL문서크기 (KB)	추론전 공리수	추론후 공리수	로딩및추론 시간(ms)
스키마	19	287	468	1011

[표 13]은 본 논문의 시스템 기반 실험 결과이다. [표 13]은 실험 DB의 스키마로부터 생성된 TBox 온톨로지를 본 논문의 시스템과 연결된 DL Reasoner를 통해서 로딩 및 초기 추론한 결과 및 시간이다. 본 논문의 시스템은 DB 인스턴스의 용량과 관계없이 [표 13]이 나타내는 초기 로딩 및 추론 시간 이후부터 온톨로지 관점에서의 DB 탐색이 가능하다.

VI. 결 론

본 논문에서 구현한 RDB-to-OWL 매핑 알고리즘은 기존의 RDB로부터 OWL 온톨로지를 생성하기 위한 기존 매핑 규칙들이 갖는 특정 구조의 테이블이 존재할 경우 OWL로의 변환이 불가능한 문제를 해결한다.

본 논문의 시스템은 DB로부터 생성된 ABox 온톨로지를 추론기에 적재하지 않으면서도 Tableau 알고리즘 기반 추론기들이 제공하는 결과와 동일한 결과를 제공할 수 있다. 이러한 처리 방식은 Tableau 알고리즘 기반 추론기들의 큰 볼륨을 갖는 ABox 온톨로지 추론에 있어서 범위성이 유연하지 못한 문제를 극복하기 위한 수단이다. 향후 연구 과제로는 사용자의 보다 자유로운 DB 기반 온톨로지 탐색을 지원하고자 OWL 지원 질의어 처리 엔진을 본 논문의 시스템 기반 하에 추가하고자 한다.

참 고 문 헌

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

[2] C. Bizer and A. Seaborne, "D2RQ - treating non-RDF databases as virtual RDF graphs," 3rd International Semantic Web Conference, 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, 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, 2007.

[5] http://www.w3.org/2001/sw/rdb2rdf/

[6] N. Cullot, R. Ghawi, and K. Yétongnon, "DB2OWL: A Tool for Automatic Database-to-Ontology Mapping," Proceedings of 15th Italian Symposium on Advanced Database System, pp.491-494, 2007.

[7] M. Li, X. Y. Du, and S. Wang, "Learning Ontology from Relational Database," Proceedings of the 4th International Conference on Machine Learning and Cybernetics, pp.3410-3415, 2005.

[8] S. S. Sane and A. Shirke, "Generating OWL ontologies from a relational databases for the semantic web," Proceedings of the International Conference on Advances in Computing, Communication and Control, pp.157-162, 2009.

[9] Z. Xu, S. Zhang, and Y. Dong, "Mapping between Relational Database Schema and OWL Ontology for Deep Annotation," Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence, pp.548-552,

2006.

[10] S. Zhou, G. Meng, H. Ling, and H. Zhang, “Tool for Translating Relational Databases Schema into Ontology for Semantic Web,” Proceedings of the 2nd International Workshop on Education Technology and Computer Science, pp.198-201, 2010.

[11] J. W. Choi and M. H. Kim, “Generationg OWL Ontology from Relational Database,” Proceedings of MUSIC'12 , pp.53-59, 2012.

[12] 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, 2004.

[13] 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, 2005.

[14] B. Motik and U. Sattler, “A Comparison of Reasoning Techniques for Querying Large Description Logic Aboxes,” Proceedings of LPAR'06, pp.227-241, 2006.

[15] http://www.w3.org/TR/owl2-syntax/

[16] http://clarkparsia.com/pellet/

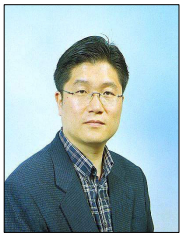
저 자 소 개

최 지 웅(Ji Woong Choi) 정회원

- 2001년 2월 : 숭실대학교 컴퓨터학부(공학사)
- 2003년 2월 : 숭실대학교 컴퓨터학과(공학석사)
- 2011년 8월 : 숭실대학교 컴퓨터학과(공학박사)
- 2013년 3월 ~ 현재 : 숭실대학교 컴퓨터학부 조교수

<관심분야> : 분산/병렬 컴퓨팅, 시맨틱 웹, IoT

김 명 호(Myung Ho Kim) 정회원



- 1989년 2월 : 숭실대학교 컴퓨터학부(공학사)
- 1991년 2월 : 포항공과대학교 전자계산학과(공학석사)
- 1995년 3월 ~ 현재 : 숭실대학교 컴퓨터학부 교수

<관심분야> : 분산/병렬 컴퓨팅, 그리드, 웹서비스, BI, 보안