JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 25 No. 10, pp. 23-34, October 2020
https://doi.org/10.9708/jksci.2020.25.10.023

Automatic Construction of SHACL Schemas for RDF Knowledge Graphs Generated by Direct Mappings

Ji-Woong Choi*

*Associate Professor, School of Computer Science and Engineering, Soongsil University, Seoul, Korea

[Abstract]

In this paper, we proposes a method to automatically construct SHACL schemas for RDF knowledge graphs(KGs) generated by Direct Mapping(DM). DM and SHACL are all W3C recommendations. DM consists of rules to transform the data in an RDB into an RDF graph. SHACL is a language to describe and validate the structure of RDF graphs. The proposed method automatically translates the integrity constraints as well as the structure information in an RDB schema into SHACL. Thus, our SHACL schemas are able to check integrity instead of RDBMSs. This is a consideration to assure database consistency even when RDBs are served as virtual RDF KGs. We tested our results on 24 DM test cases, published by W3C. It was shown that they are effective in describing and validating RDF KGs.

▸ **Key words**: Knowledge graph, RDF, SHACL, Direct Mapping, RDF validation

[요 약]

본 논문에서는 Direct Mapping(DM) 방식으로 생성된 RDF 지식 그래프에 대한 SHACL 스키마를 RDB 스키마로부터 자동 생성하는 방법을 제안한다. DM과 SHACL은 모두 W3C 표준 사양이다. DM은 RDB 데이터를 RDF 그래프로 변환하기 위한 규칙들로 구성되어 있다. SHACL은 RDF 그래프의 구조 묘사와 구조 검증을 위한 언어이다. 제안하는 방법은 RDB 스키마의 구조 정보뿐 아니라 무결성 제약조건을 SHACL로 자동 번역한다. 즉, 자동 생성된 SHACL 스키마는 RDBMS를 대신하여 무결성 제약조건 위배 여부를 검증할 수 있다. 이것은 RDB가 RDF 표현의 가상 지식 그래프로서 서비스되는 상황에서도 데이터베이스의 일관성을 보장하기 위한 고려이다. 자동 생성된 SHACL 스키마를 W3C가 발표한 24가지 DM 테스트 케이스에 적용하여 RDF 그래프의 구조 설명과 검증에 있어서 유효함을 보였다.

▸ **주제어**: 지식 그래프, RDF, SHACL, 직접 사상, RDF 검증

• First Author: Ji-Woong Choi, Corresponding Author: Ji-Woong Choi
*Ji-Woong Choi (iamjwchoi@gmail.com), School of Computer Science and Engineering, Soongsil University
• Received: 2020. 09. 03, Revised: 2020. 09. 21, Accepted: 2020. 09. 22.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

지식 그래프는 도메인 지식을 그래프 형식으로 구조화하여 웹에서 공유할 목적으로 공개한 데이터 세트를 의미한다[1]. 시맨틱 웹을 위해 정의한 W3C 표준 기술들은 오늘날 지식 그래프 구축을 위한 사실상의 표준으로 사용되고 있다[2]. 데이터 표현 및 교환 용도의 RDF(Resource Description Framework)와 데이터 조회 및 조작 용도의 SPARQL 질의어가 그 예이다. RDF 지식 그래프의 활용성을 높이기 위해 극복해야 할 문제 중 하나는 RDF 그래프를 대상으로 한 질의-응답 과정의 비효율성이다[3]. 즉, SPARQL 사용자는 여러 번의 질의-응답 과정을 거친 후에야 원하는 데이터를 획득할 수 있는 최종 질의문을 결정할 수 있게 된다. 하지만 사용자는 그 결정에 대하여 확신할 수 없다[4]. 이러한 비효율성을 초래하는 원인 중 하나는 RDF 그래프에 대한 구조 정보가 사용자에게 제공되지 않았기 때문이다[5]. 이것은 SQL 사용자에게 RDB 스키마 정보가 제공되지 않은 상황에 비견될 수 있다. 이 문제는 근본적으로 RDF 모델이 스키마 기반의 관계형 모델과는 달리 본래 스키마 없는(schema-free) 데이터 모델이라는 특성에 기인한다. 즉, RDF는 구조에 대한 정의 없이도 구축 가능한 유연한 데이터 모델이기 때문이다. 그러나 현실에서의 지식 그래프가 효용을 높이기 위하여 구조적으로 구축됨으로 인해 구조 정보를 표현하고 교환하기 위한 표준 형식에 대한 필요가 높아져 왔다. 이에 W3C는 RDF 그래프의 구조를 묘사하고 검증할 수 있는 표준 사양 SHACL(Shapes Constraint Language)[6]을 2017년 발표하였다.

RDF 지식 그래프는 다양한 데이터 원천으로부터 구축될 수 있다. 근래 들어 기계 학습 기술의 진보에 힘입어 비정형 데이터로부터 신뢰성 있는 지식 그래프를 구축하기 위한 연구가 활발히 수행되고 있다[7]. 이러한 연구들은 공정상의 효율은 가져왔으나 데이터 품질 측면에서는 아직 만족할만한 결과를 보이지 못하고 있다[8]. 반면, 정형데이터로부터 고품질의 RDF 그래프를 구축하는 것은 상대적으로 용이할 뿐 아니라 제반 기술 또한 성숙하다. 현재까지도 RDF 그래프 구축에 사용되는 지배적인 위치의 정형 데이터 원천은 관계형 데이터베이스(RDB)다[9]. RDB 데이터를 RDF 그래프로 변환하는 방식은 디폴트 매핑(default mapping)과 커스텀 매핑(customized mapping)으로 분류할 수 있다. 이들 각각에 대한 W3C 표준 또한 이미 발표되어 있다. Direct Mapping (DM)[10]은 디폴트 매핑에 대한 표준이며 R2RML[11]은 커스텀 매핑에 대한 표준이다. DM은 RDB 데이터 전체를 관계형 모델의 의미에 따라 RDF 그래프 표현으로 변환시키기 위한 규칙의 모음이다. DM은 주로 가상 지식 그래프를 위해서 사용된다[12]. 가상 지식 그래프는 SPARQL 사용자와 RDBMS 사이에서 RDBMS가 RDF 저장소로 보이게 하는 가상화 시스템의 중계를 통해 서비스된다. 이 시스템은 사용자로부터의 SPARQL 요청을 SQL로 변환하여 RDBMS로 전달하며 돌려받은 RDBMS로부터의 결과 데이터만을 대상으로 마치 DM에 의해 미리 생성해둔 그래프에서의 결과와 같도록 즉시 RDF로 변환하여 사용자에게 돌려준다. R2RML은 변환 규칙을 자유롭게 정의할 수 있는 구문을 제공하는 언어다. R2RML로 작성된 매핑 규칙에 따라 R2RML 프로세서가 변환을 수행한다. R2RML에 의해 생성된 RDF 그래프는 DM에 의해 생성된 RDF 그래프와는 달리 통상 그래프 전용 저장소에 물리적으로 저장되어 서비스된다. DM과 R2RML 모두 RDB로부터 데이터 그래프만을 생성할 뿐 그 그래프에 대한 스키마 생성은 사양에 포함되지 않는다.

본 논문에서는 DM에 의해 생성된 RDF 그래프에 대한 SHACL 스키마를 RDB 스키마로부터 자동 생성하는 방법을 제안한다. 제안하는 방법은 DM이 데이터 그래프 생성만을 목적으로 하기에 묵시적으로 반영된 무결성 제약조건조차 RDB 스키마에서 추출하여 SHACL 스키마에 그 의미를 명시적으로 반영시킨다. 이것은 RDB가 DM을 따른 가상 지식 그래프로서 접근되는 상황에서도 무결성이 유지되도록 하기 위한 고려다. 즉, SHACL 스키마를 준수할 경우 RDB 무결성 제약조건 또한 만족하게 하기 위함이다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구와 배경지식을 기술한다. 3장에서는 RDB 스키마로부터 SHACL 스키마를 자동생성하는 과정과 방법을 기술한다. 4장에서는 제안하는 방법을 구현한 시스템의 테스트 결과를 제시하며, 5장에서는 결론을 맺는다.

II. Preliminaries

1. Related works

이 절에서는 SHACL 개발 이전에 시맨틱 웹 표준 기술을 활용하여 RDF 그래프 구조 묘사 및 검증을 위해 시도된 연구 사례를 분석한다. 이 목적을 위해 전통적으로 사용된 언어는 OWL(Web Ontology Language)[13]이다. OWL은 서술 논리(Description Logic) 기반의 언어다. 따라서 OWL 기반의 접근법은 관계형 모델의 의미를 OWL 공리로 번역한다. 연구 [14]는 DM 명세의 공저자 중 하나인 J. Sequeda의 연구로서 이 접근법 초기의 대표 사례이며 DM에 충실한 OWL 스키마를 생성한다. 즉, DM에서

명시적으로 반영한 무결성 제약조건만 OWL 공리로 명시적으로 표현하고 그렇지 않은 Primay Key, Not Null, Check 제약조건은 표현되지 않았다. 연구 [15]는 연구 [14]의 한계가 극복된 비교적 최근의 대표 사례로서 연구 [14]에서 누락된 제약조건도 OWL 공리로 표현했을 뿐만 아니라 그래프 버전 관리, 그래프 병합 시 활용될 수 있도록 provenance 메타 정보를 OWL 온톨로지에 포함 시킨 것이 특징이다.

OWL을 사용할 때의 한계는 두 가지를 꼽을 수 있다. 첫째, 구조 검증 관점에서의 불완전성이 존재한다[16]. OWL을 사용한 그래프 구조 검증은 추론기에 의해 수행된다. 추론기의 본래 목적은 구조 검증이 아닌 추론 즉 새로운 지식 발견에 있다. 다만, ABox 추론 중 발견된 논리적 모순을 제약조건 위반으로 해석하여 활용할 뿐이다. 이 과정에서 열린계 가정 즉 참이라고 알려지지 않은 것은 거짓이 아닌 모름으로 판단하는 방식을 취하는 OWL의 특성으로 인해 구조적으로 무결성 위반인 인스턴스가 논리적 모순은 아닌 경우 검출되지 못하는 문제가 발생한다. 둘째, OWL은 RDF 그래프에 대한 직관적인 구조 정보를 SPARQL 사용자에게 제공하지 못한다. SPARQL은 OWL이 아닌 RDF를 위한 언어다. OWL은 논리 기반의 지식 표현 언어다. 반면, SPARQL은 그래프 패턴 매칭 방식의 질의어다. 이 갭에 의해 SPARQL 사용자는 OWL로 명세된 논리로부터 RDF 그래프 패턴을 유추해야 하는 어려움과 불편함을 감내해야 한다. 이러한 간극을 해소하기 위해 RDF 그래프를 대상으로 OWL 논리 관점으로 질의가 가능한 SQWRL[17], SPARQL-OWL[18], SPARQL-DL[19] 등의 질의어가 제안되었으나 널리 사용되고 있지는 않다.

2. Review of Direct Mapping

본 절에서는 DM이 RDB로부터 RDF 그래프를 생성하는 방법을 기술한다. 표 1은 RDB 스키마 요소로부터 RDF 요소를 생성하는 규칙을 요약한다. RDB 스키마에 정의된 각각의 테이블, 컬럼, 외래키로부터 IRI를 하나씩 생성한다. DM은 무슨 RDB 스키마 요소로부터 생성된 IRI 인지에 따라 table IRI, literal property IRI, reference property IRI로 구분하며 이 명칭은 DM에서만 유효하다. 표 2는 RDB 인스턴스 요소로부터 RDF 요소를 생성하는 규칙을 요약한다. DM은 기본적으로 RDB에 존재하는 모든 테이블 내의 모든 컬럼값으로부터 RDF literal을 각각 생성한다. RDF literal은 값과 데이터 타입으로 구성된다. 따라서 RDF literal 생성을 위해서는 사실 컬럼값 뿐만 아니라 컬럼의 데이터 타입 정보가 함께 사용된다. 이 두 정보를 바탕으로 DM이 RDF literal을 구성하는 규칙은 R2RML 사양에서 정의한 SQL 값과 RDF literal 간의 매핑 규칙을 그대로 따른다. DM은 추가적으로 RDB에 존재하는 모든 테이블 내의 모든 행으로부터 row node라는 것을 각각 생성한다. 기본키가 정의된 테이블의 행이면 IRI인 row node를 생성하며 기본키가 정의되지 않은 테이블의 행은 blank node인 row node를 생성한다.

Table 1. Mappings from RDB Schema

RDB Schema	DM	RDF
table	table IRI	IRI
column	literal property IRI	IRI
foreign key	reference property IRI	IRI

Table 2. Mappings from RDB Instance

RDB Instance	DM	RDF
row	row node	IRI or blank node
column value	RDF literal	RDF literal

DM은 DM에서의 IRI 구분에 따른 IRI 문자열 포맷을 각각 정의하고 있다. 이 포맷은 생성되는 IRI가 고유할 수 있도록 대응되는 RDB 스키마 요소의 이름 혹은 RDB 인스턴스의 컬럼값이 포맷 내의 정해진 위치에 포함되도록 한다. 구체적으로 table IRI는 테이블명을 포함한다. literal property IRI는 테이블명과 컬럼명을 포함한다. reference property IRI는 외래키를 갖는 테이블명, 외래키 컬럼명 목록을 포함한다. row node는 테이블명, 기본키 컬럼명과 컬럼값의 쌍 목록을 포함한다. DM은 복수의 컬럼 목록이 IRI에 포함될 경우 SQL DDL 문의 키 정의에서의 컬럼명 나열 순서를 따르도록 한다. 그리고 DM은 blank node의 작명 규칙은 정의하지 않고 있다. 그 이유는 RDF 사양이 blank node에 대해서 로컬 즉 해당 RDF 그래프 내에서 유일하게 식별되도록 하는 기능만을 요구하기 때문이다. 즉, RDF 응용이 RDF 그래프 내의 blank node의 이름을 변경해도 그 기능이 유지된다면 RDF 사양에 위배되는 사용은 아니다.

표 3은 DM에 의해 생성되는 세 가지 유형의 트리플을 정리한 것이며 그림 1은 예시 RDB로부터 DM에 의해 생성된 RDF 그래프를 보여준다.

Table 3. RDF Triple Patterns on Direct Mapping

subject	predicate	object
row node	rdf:type	table IRI
row node	literal property IRI	RDF literal
row node	reference property IRI	row node

Tasks

		PK		
Projects			→ Projects(name, deptName)	
name	**deptName**	**id**	**project**	**deptName**
pencil	accounting	7	pencil	accounting

```
01 @base <http://foo.example/DB/>
02 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
03 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
04
05 _:c rdf:type <Projects> .
06 _:c <Projects#name> "pencil" .
07 _:c <Projects#deptName> "accounting" .
08
09 <Tasks/id=7;project=pencil> rdf:type <Tasks> .
10 <Tasks/id=7;project=pencil> <Tasks#id> 7 .
11 <Tasks/id=7;project=pencil> <Tasks#project> "pencil" .
12 <Tasks/id=7;project=pencil> <Tasks#deptName> "accounting" .
13 <Tasks/id=7;project=pencil> <Tasks#ref-project;deptName> _:c .
```

Fig. 1. An Example of Direct Mapping

그림 1의 5~7행은 Project 테이블에 제시된 1개의 행으로부터 생성되었으며 9~13행은 Tasks 테이블에 제시된 1개의 행으로부터 생성되었다. DM은 표 3의 1행 유형을 row type triple 2행 유형을 literal triple 3행 유형을 reference triple로 분류한다. 그림 1에서 5행과 9행이 row type triple의 예다. 6~7행과 10~12행이 literal triple의 예다. 13행이 reference triple의 예다. row type triple은 테이블과 그 테이블 소속 행 사이의 관계를 RDF 그래프로 반영한다. 생성된 그래프에서 테이블에 대응하는 table IRI는 그 테이블 소속 행에 대응하는 row node들의 공통 타입이 된다. literal triple은 행과 그 행 소속 컬럼값 사이의 관계를 RDF 그래프로 반영한다. 생성된 그래프에서 행에 대응하는 row node는 그 행 소속 컬럼값에 대응하는 RDF literal들의 공통 주어가 된다. reference triple은 외래키로 인해 두 행 사이에 성립된 참조 관계를 RDF 그래프로 반영한다. 생성된 그래프에서 참조하는 행에 대응하는 row node가 주어, 외래키에 대응하는 reference property IRI가 술어, 참조 당하는 행에 대응하는 row node가 목적어가 된다.

III. The Proposed Method

1. System Overview

그림 2는 세 개의 계층으로 구성된 제안하는 시스템의 구조를 보여준다.

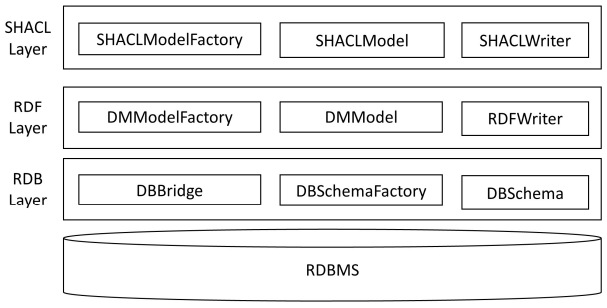

Fig. 2. Architecture of the Proposed System

RDB 계층은 RDBMS에 접근하여 RDB 스키마 정보와 RDB 인스턴스 데이터를 수집하여 상위 계층에 공급하는 기능을 담당한다. 직접적인 RDBMS로의 접근은 DBBridge가 모두 담당한다. 시스템은 RDF 그래프와 SHACL 스키마를 함께 생성할 수도 있으며 각각을 독립적으로 생성할 수도 있도록 설계되었다. 어느 경우에나 RDB 스키마 정보가 공통적으로 사용되며 RDF 그래프 생성이 포함된 경우에만 RDB 인스턴스 데이터가 함께 사용된다. 따라서 시스템은 RDBMS와의 연결이 수립되면 RDF 혹은 SHACL 모델 생성 이전에 DBSchemaFactory를 실행시켜 RDB 스키마 정보를 DBBridge를 통해 일괄적으로 수집하여 DBSchema를 로컬에 구축해 둔다. 이후 상위 계층은 DBSchema의 인터페이스를 통해 RDB 스키마 정보를 공급받는다.

RDF 계층은 DM에 의한 RDF 그래프 생성을 담당한다. DM 매핑 규칙은 RDFModelFactory와 RDFWriter에 구현되어 있다. RDFModelFactory에는 RDB 요소로부터 RDF 요소를 생성하는 규칙이 RDFWriter에는 RDF 요소를 재료로 트리플을 구성하는 규칙이 구현되어 있다. RDFModel은 RDFModelFactory가 구축한다. RDFModel은 RDB 스키마 요소로부터 생성된 완성된 형태의 RDF 요소와 구체적인 RDB 인스턴스 데이터가 공급되면 완성된 RDF 요소를 반환해 주는 템플릿 역할로서만 존재하는 미완성 RDF 요소로 구성되어 있다. 완성된 형태의 RDF 요소는 각각 자신에 대응하는 RDB 스키마 요소를 보관하고 있다. RDFWriter는 RDF 그래프 출력을 담당한다. RDFWriter는 RDFModel에 존재하는 RDF 요소를 트리플로 조립하여 완성된 RDF 그래프를 출력한다. 이때, RDFWriter는 RDFModel에 있는 템플릿 형태의 미완성 요소가 제공하는 함수에 DBBridge를 통해 수급한 RDB 인스턴스 데이터를 인자로 건네 호출한 후 완성된 형태의 요소를 반환받는 과정을 밟는다. 즉, RDFModel은 언제나 개별 RDB 인스턴스 요소 각각에 대응하는 RDF 요소를 담지 않는다.

SHACL 계층은 DM에 의해 생성된 RDF 그래프에 대한 SHACL 스키마 생성을 담당한다. SHACLModelFactory

는 RDFModel로부터 SHACLModel을 생성한다. SHACLModel의 구성 요소는 shape이다. shape의 구성 요소는 conststraint이다. 각각의 shape은 생성 시 전달된 RDF 요소 뿐만 아니라 그 RDF 요소에 보관되어 있는 대응하는 RDB 스키마 요소 둘 다로부터 부속 constraint를 구성하기 위해 필요한 값을 구해둔다. SHACLWriter는 SHACLModel의 각 요소를 SHACL 문법에 맞추어 문서 형태로 출력한다.

2. Extraction of Relational Schema

그림 3은 RDF 모델과 SHACL 모델 생성의 기반이 되는 DBSchema로 수집된 RDB 스키마 정보의 구조를 보여준다. 그림 3에서 볼드체 박스는 클래스이고 이탤릭체 박스는 속성이다.

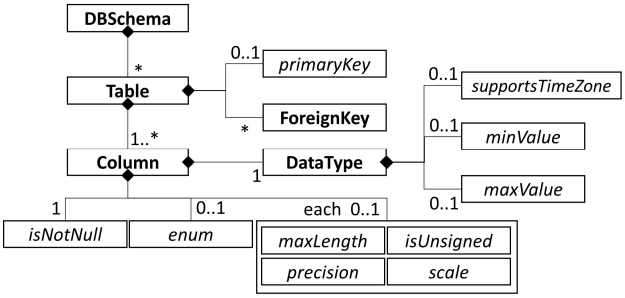

Fig. 3. Structure of DBSchema

DBSchema는 테이블의 집합이다. 테이블은 컬럼 목록과 기본키 속성 그리고 외래키 목록으로 구성된다. 기본키와 외래키는 키 정의를 기반으로 키를 구성하는 컬럼 목록의 순서를 보존한다. 외래키는 추가로 키 정의에 포함된 두 테이블의 컬럼간 대응 관계도 보존한다. 컬럼은 반드시 그 컬럼의 데이터 타입과 널값 허용 여부 정보를 갖고 있다. 데이터 타입은 정수, 날짜, 시간 타입에 한하여 그 타입에서 허용된 최솟값과 최댓값을 보관한다. 이 값들은 데이터 타입 자체의 성질로서 RDBMS마다 그 범위가 다르다. 또한, 날짜와 시간 타입의 경우 RDBMS마다 타임존을 지원하는 타입과 그렇지 않은 타입으로 구분되어 제공되므로 이 성질 또한 속성으로서 보관한다. 컬럼에는 컬럼값의 범위에 대한 열거형의 제약이 존재할 수 있다. 그림 3의 *enum*은 이 제약에 열거된 후보를 보관하는 속성이다. 문자열 타입의 경우 컬럼마다 최대 문자수를 달리 지정할 수 있다. 그림 3의 *maxLength*는 이 값을 저장한다. 숫자 타입에 한하여 음수 저장을 금지시킨 컬럼이 존재할 수 있다. 그림 3의 *isUnsigned*는 이 조건의 명세 여부를 저장한다. 그림 3의 *precision*과 *scale*은 고정 소수점 숫자 타입의 컬럼마다 달리 정해질 수 있는 전체 자릿수와 소수점 이하 자릿수 정보를 저장한다. *precision*은 시간 타입의 컬럼에 대해서도 사용된다. 이때는 마이크로초 표현부에 대한 자릿수를 의미한다.

3. Construction of RDF Model

그림 4는 RDFModelFactory가 DM의 매핑 규칙에 따라 DBSchema로부터 구축한 RDFModel의 구조를 보여준다. 그림 4에서 볼드체 박스는 클래스이고 이탤릭체 박스는 변수명이다. RDFModel의 속성 *baseIRI*는 구축될 그래프에서 RDB 요소로부터 생성되는 모든 IRI에 대한 공통된 접두어로서 RDFModel 객체 생성 시 외부에서 주어진다. RDFModel을 제외한 나머지 클래스는 모두 해당하는 RDF 요소 생성 시 필요한 DM의 작명 규칙을 각각 내장하고 있다.

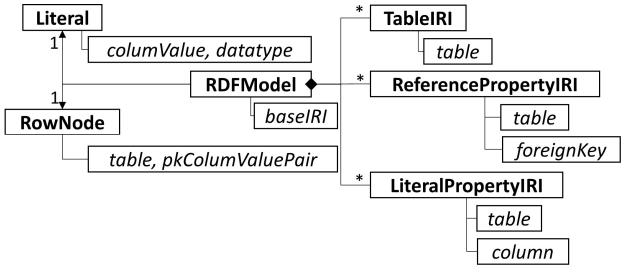

Fig. 4. Structure of RDFModel

그림 4에서 RDFModel 우측에 위치한 세 개의 클래스는 RDB 스키마 요소에 1:1 대응하는 객체 형태의 RDF 요소를 생성하며 생성된 객체들은 RDFModel에서 유지된다. 각각의 객체는 RDFWriter의 요청 시 자신을 IRI 형식으로 출력하는 기능을 갖추고 있다. 세 개의 클래스 하위의 속성들은 각각 객체 생성 시 건네진 RDB 스키마 요소명이다. 이 속성들은 기본적으로 해당 IRI 완성을 위해 필요한 값들이지만 SHACL 계층에서 묘사하고자 하는 RDF 요소를 생성케 한 RDB 요소의 메타 정보를 DBShema에서 찾을 때 검색키로 사용된다.

그림 4에서 Literal과 RowNode 클래스는 RDB 인스턴스 요소에 대응하는 RDF 요소를 생성하는 기능만을 제공한다. 즉, 두 클래스로부터 RDFModel에서 유지되는 객체는 따로 생성되지 않는다. RDF 계층에서 두 클래스에 대한 사용자는 RDFWriter다. 그림 4에서 Literal과 RowNode 클래스의 하위에 있는 변수 목록은 RDFWriter가 RDF 요소 획득을 위해 두 클래스가 제공하는 기능에 전달해야 하는 매개변수 목록이다.

4. Construction of SHACL Schema

그림 5는 SHACLModel의 구조를 보여준다. 그림 5의 모든 박스는 클래스명이다.

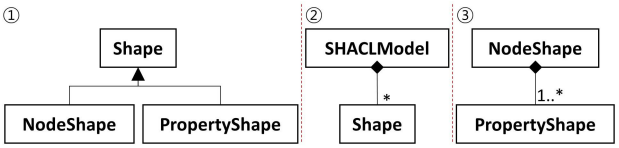

Fig. 5. Structure of SHACLModel

SHACL 사양은 SHACL 논리 모델을 구성하기 위한 단위로서 Shape 클래스를 정의하고 있으며 Shape 클래스에 대한 서브클래스로서 NodeShape와 PropertyShape를 정의하고 있다. 그림 5의 ①은 시스템이 이 상속 관계를 그대로 반영했음을 보인다. 그림 5의 ② 또한 시스템이 SHACL 논리 모델에 해당하는 SHACLModel을 Shape 객체의 집합이 되도록 설계하였음을 보인다. SHACLModel 객체가 실제로 포함하는 객체의 타입은 NodeShape 혹은 PropertyShape 둘 중 하나지만 ①의 상속 관계를 활용하여 모두 Shape로서 참조하도록 단순화시켰다.

SHACL은 NodeShape와 PropertyShape에 대한 역할 배분을 NodeShape가 트리플에서 주어 노드 혹은 RDF literal이 아닌 목적어 노드에 대한 형상 묘사를 담당토록 했으며 PropertyShape가 트리플에서 술어와 목적어의 쌍 혹은 주어와 술어의 쌍에 대한 형상 묘사를 담당토록 했다. 시스템은 이 역할 배분을 따른다. 표 3은 시스템이 묘사해야 하는 세 가지 트리플 유형이다. 시스템의 NodeShape는 표 3의 세 가지 트리플 유형에 출현하는 모든 row node의 묘사를 담당한다. 시스템의 PropertyShape는 표 3의 literal triple에서의 술어와 목적어 쌍의 묘사와 reference triple에서의 주어와 술어 쌍 그리고 술어와 목적어 쌍의 묘사를 담당한다. 마지막으로 row type triple에서의 술어와 목적어 쌍은 시스템의 NodeShape가 PropertyShape의 도움 없이 묘사한다. 그 이유는 SHACL이 노드 타입에 대한 제약조건을 NodeShape에 할당했기 때문이다.

Table 4. Mappings from RDF Model

SHACL Model	RDF Model
NodeShape	TableIRI
[Type1]PropertyShape	LiteralPropertyIRI
[Type2, Type3]PropertyShape	ReferencePropertyIRI

표 4는 SHACLModelFactory에 대한 입력으로서의 RDFModel 요소와 출력으로서의 SHACLModel 요소 간 대응 관계를 정리한 것이다. 시스템은 TableIRI 객체 1개당 NodeShape 객체 1개를 생성한다. NodeShape 객체 1개는 1개의 테이블에서 생성된 모든 row node의 묘사를 담당한다. 시스템은 LiteralPropertyIRI 객체 1개당 PropertyShape 객체 1개를 생성한다. 이 경우의 PropertyShape를 Type1이라 구분하겠다. Type1 PropertyShape 객체 1개는 1개의 컬럼에서 생성된 literal triple 유형의 모든 술어와 목적어 쌍의 묘사를 담당한다. 시스템은 ReferencePropertyIRI 객체 1개당 PropertyShape 2개를 생성한다. 이 경우의 PropertyShape 객체들은 1개의 외래키 정의로 인해 reference triple 유형으로 연결된 모든 row node 쌍의 참조 관계를 묘사한다. 다만 하나는 주어 row node를 위한 술어와 목적어 쌍의 묘사용도이며 이를 Type2라 하겠다. 다른 하나는 목적어 row node를 위한 주어와 술어 쌍의 묘사용도이며 이를 Type3라 하겠다.

그림 5의 ③은 NodeShape가 PropertyShape 여럿을 참조하게 됨을 의미한다. 시스템은 PropertyShape 객체를 생성 후 SHACLModel에도 등록하지만 NodeShape에도 등록한다. PropertyShape 객체는 자신이 묘사하는 트리플의 두 요소를 제외한 나머지 한 요소를 묘사하는 NodeShape 객체에 등록된다. 이러한 관계 설정 방식으로 시스템은 모든 NodeShape 객체가 자신의 묘사 대상과 관련된 모든 제약조건을 소유하게 한다.

그림 6~9는 SHACLModel 내의 각각의 Shape가 그 종류에 따라 출력하게 되는 SHACL 구문이다. 그림 6~9에서 볼드체는 SHACL이 정의한 제약조건이며 각 행 우측의 '1', '?', '+', '*'는 해당 행의 제약조건이 출력될 수 있는 횟수를 뜻한다. '1'은 1회, '?'은 0 또는 1회, '+'은 1회 이상, '*'은 0회 이상이다.

```
01  nodeShapeIRI                                              1
02    a sh:NodeShape ;                                        1
03    sh:targetClass tableIRI ;                               1
04    sh:nodeKind sh:IRI | sh:BlankNode ;                     1
05    sh:class tableIRI ;                                     1
06    sh:pattern regExp ;                                     ?
07    sh:property [Type1]propertyShapeIRI ;                   +
08    sh:property [Type2]propertyShapeIRI ;                   *
09    sh:property [Type3]propertyShapeIRI .                   *
```

Fig. 6. NodeShape Syntax

01	[Type1]propertyShapeIRI	1
02	a sh:PropertyShape ;	1
03	**sh:path** literalPropertyIRI ;	1
04	**sh:nodeKind** sh:Literal ;	1
05	**sh:datatype** IRI ;	1
06	**sh:in** (literalSet) ;	?
07	**sh:pattern** regExp ;	?
08	**sh:minInclusive** literal ;	?
09	**sh:maxInclusive** literal ;	?
10	**sh:maxLength** nonNegativeInteger ;	?
11	**sh:minCount** 1 ;	?
12	**sh:maxCount** 1 .	1

Fig. 7. Type1 PropertyShape Syntax

01	[Type2]propertyShapeIRI	1
02	a sh:PropertyShape ;	1
03	**sh:path** referencePropertyIRI ;	1
04	**sh:nodeKind** sh:IRI | sh:BlankNode ;	1
05	**sh:class** tableIRI ;	1
06	**sh:minCount** 1 ;	?
07	**sh:maxCount** 1 .	1

Fig. 8. Type2 PropertyShape Syntax

01	[Type3]propertyShapeIRI	1
02	a sh:PropertyShape ;	1
03	**sh:path** [**sh:inversePath** referencePropertyIRI] ;	1
04	**sh:nodeKind** sh:IRI | sh:BlankNode ;	1
05	**sh:class** tableIRI .	1

Fig. 9. Type3 PropertyShape Syntax

그림 6~9의 1행은 각각의 Shape에 대한 식별자 IRI이며 2행에서 1행의 IRI가 무슨 종류의 Shape인지를 밝힌다. 그림 6~9의 3행부터는 Shape의 종류에 따라 사용하는 제약조건을 달리한다.

NodeShape을 구성하는 제약조건의 값은 table IRI로부터 획득한 원천 테이블의 스키마 정보를 바탕으로 생성한다. 그림 6의 sh:targetClass는 검증 대상 노드를 지정한다. NodeShape마다 생성 시 전달된 table IRI를 값으로 사용한다. 이것은 NodeShape마다 각기 다른 테이블에서 생성된 row node 모두를 검증 대상으로 삼게 한다. sh:nodeKind는 검증 대상의 종류를 제한한다. 원천 테이블에 기본키가 정의되어 있으면 sh:IRI가 없다면 sh:BlankNode가 값이 된다. sh:IRI는 노드가 IRI여야 함을 sh:BlankNode는 노드가 blank node여야 함을 강제한다. sh:class는 검증 대상이 특정 타입의 인스턴스여야 함을 의미한다. 이 조건은 검증 대상이 row type triple에서 주어이고 그 트리플의 목적어가 sh:class의 값과 같다면 만족된다. sh:pattern은 검증 대상의 IRI 문자열이 부합해야 하는 정규 표현식을 값으로 취한다. 이 조건은 sh:nodeKind의 값이 sh:IRI일 경우에만 출력된다. 이 조건의 값은 테이블명과 기본키 컬럼목록을 매개로 한 RowNode 클래스의 기능 호출을 통해서 획득한다. sh:property는 검증 대상 노드가 충족시켜야 하는 트리플 구조에 대한 제약조건을 값으로 취한다. sh:property의 출력 횟수는 해당 NodeShape 객체가 참조하는 PropertyShape 객체의 개수와 같으며 sh:property의 값으로서 각각의 PropertyShape 객체에 대응하는 properyShapeIRI가 사용되며 이는 참조를 의미한다. 그림 6의 7행의 구문은 원천 테이블의 컬럼 개수만큼 출력되며 8행의 구문은 원천 테이블에 정의된 외래키 개수만큼 출력되며 9행의 구문은 원천 테이블을 참조하는 외래키 개수만큼 출력된다.

Table 5. Constraints Used Depending on Data Type

Data Type	Used Constraint
xsd:boolean	sh:in
xsd:date, xsd:dateTime	sh:pattern sh:minInclusive sh:maxInclusive
xsd:decimal, xsd:time	sh:pattern
xsd:double	sh:minInclusive
xsd:hexBinary, xsd:string	sh:maxLength
xsd:integer	sh:minInclusive sh:maxInclusive

Type1 PropertyShape을 구성하는 제약조건의 값은 literal property IRI로부터 획득한 원천 컬럼의 제약조건을 바탕으로 생성한다. 그림 7의 sh:path는 값으로 할당된 literalPropertyIRI로 술어를 특정한다. sh:nodeKind의 값은 sh:Literal로 고정된다. 그 이유는 Type1 PropertyShape가 literal triple의 술어와 목적어를 묘사하기 때문이다. sh:datatype은 목적어의 데이터 타입을 sh:datatype의 값으로 제한한다. sh:datatype의 값은 원천 컬럼의 데이터 타입에 대응하는 RDF literal 데이터 타입이다. SHACL에서는 RDF literal 데이터 타입으로서 XML 스키마 언어[20]가 정의한 IRI를 사용한다. sh:in의 값은 RDF literal의 집합이다. 집합의 원소는 그림 3의 *enum* 속성 항목 각각을 RDF literal들로 변환한 것이다. sh:in으로 인해 목적어 범위가 이 집합의 원소로 제한된다. sh:pattern은 목적어가 준수해야 하는 문자열 패턴을 정의한다. sh:pattern은 표 5에서 제시된 바와 같이 데이터 타입이 xsd:date, xsd:dateTime, xsd:decimal, xsd:time일 때만 사용된다. 날짜와 시간 관련 타입의 경우 RDB에서의 값 표현 형식과 XML 스키마 데이터 타입에서 요구하는 형식이 다를 때가 있기 때문에 이 제약조건으로 올바른 형식 변환이 이루어졌는지를 점검할 수 있다. SQL

데이터 타입 중 xsd:decimal로 매핑되는 타입의 컬럼은 *precision*과 *scale*을 그리고 시간 표현이 포함된 컬럼은 *scale*을 컬럼마다 달리 갖을 수 있다. 따라서 이러한 조건의 의미를 정규 표현식에 반영하였다. sh:minInclusive와 sh:maxInclusive는 최솟값과 최댓값을 의미한다. 표 5에서 이 제약조건을 사용하는 데이터 타입들은 각각 다수의 서로 다른 값의 범위를 갖는 SQL 데이터 타입과 매핑된다. 따라서 원천 컬럼의 값의 범위를 이 제약조건에 반영하였다. xsd:double은 최솟값 제약조건만을 사용한다. xsd:double로 매핑되는 SQL 데이터 타입의 컬럼은 *unsigned* 속성이 추가될 수 있다. 따라서 이 때의 최솟값 0을 표현하기 위함이다. sh:maxLength는 컬럼값 내의 최대 허용 문자수이다. xsd:string과 xsd:hexBinary로 매핑되는 SQL 데이터 타입들 또한 컬럼마다 이 조건을 달리할 수 있으며 이 조건을 sh:maxLength에 반영하였다. sh:minCount와 sh:maxCount는 해당 PropertyShape을 만족시키는 술어와 목적어 쌍이 주어 노드에 몇 개 연결될 수 있는가를 결정한다. 이 조건은 원천 컬럼의 *isNotNull* 속성값에 의해 null을 허용하는 컬럼이면 최대 1개가 되며 null을 허용하지 않는 컬럼이면 반드시 1개가 된다. 그 이유는 DM이 null을 RDF literal로 변환하지 않기 때문이다.

Type2와 Type3 PropertyShape 쌍을 구성하는 제약조건의 값은 동일한 reference property IRI로부터 획득한 원천 외래키 정의를 바탕으로 생성한다. 그림 8~9의 sh:path는 값으로 할당된 referencePropertyIRI로 술어를 특정한다. 다만 그림 9의 sh:inversePath 제약조건에 의해 해당 PropertyShape을 sh:property로 참조하는 NodeShape의 검증 대상 노드는 검증기에 의해 특정된 술어의 목적어로 해석된다. 따라서 그림 9의 4~5행은 주어 노드에 대한 제약조건으로 해석된다. 반면에 그림 8의 4~5행은 목적어 노드에 대한 제약조건으로 해석된다. SHACL에서는 sh:minCount를 생략한 경우 최소 0회로 해석한다. 따라서 그림 8은 해당 제약조건을 만족하는 술어와 목적어 쌍이 최대 1개 가능함을 의미하며 그림 9는 해당 제약조건을 만족하는 주어와 술어 쌍의 개수에 대한 제한이 없음을 의미한다. 그 근거는 RDB에서 외래키를 갖는 행은 1개의 행만을 참조할 수 있지만 반대로 외래키에 의해 참조되는 행은 자신을 참조하는 다른 여러 행을 가질 수 있기 때문이다. 외래키를 소유한 행이 다른 행을 참조하지 않을 수도 있으며 이 경우는 외래키 컬럼값들 중 null이 포함된 때다. 따라서 외래키 컬럼들 중 null을 허용하는 컬럼이 존재하면 그림 8의 6행 출력은 생략된다.

IV. Experiments

시스템의 유효성을 검증하기 위해 W3C의 “R2RML and Direct Mapping Test Cases”[21]을 활용하였다. 이 문서의 본래 검증 대상은 R2RML 혹은 DM 프로세서다. 즉, 프로세서가 타겟 매핑 사양을 어느 정도 충실히 구현했는지를 평가하기 위함이다. 이 문서는 87개의 테스트 케이스를 포함하며 이들을 26개의 범주로 나누었다. 범주마다 RDB 구조를 달리한다. DM을 위해서는 24개 그리고 R2RML을 위해서는 63개의 테스트 케이스를 할애했다. 범주명은 D000~D025이며 각각의 범주에는 DM과 R2RML을 위한 테스트 케이스가 혼재한다. 본 테스트에서는 DM을 위한 테스트 케이스만을 활용했다. 각각의 테스트 케이스의 구성 요소는 RDB 생성을 위한 SQL 스크립트와 프로세서가 생성해야 할 RDF 문서다. 테스트는 RDF 테스트와 SHACL 테스트로 구성하였다. RDF 테스트는 문서 [21]의 본래 취지에 부합하는 테스트로서 주어진 SQL 스크립트로 생성한 RDB로부터 시스템의 RDFWriter가 RDF 문서를 출력토록 한다. 출력된 RDF 문서가 테스트 케이스에 제시된 RDF 문서와 같으면 통과로 판정하였다. SHACL 테스트는 시스템의 SHACLWriter가 SHACL 문서를 출력토록 한 후 출력된 문서와 테스트 케이스에서 제시한 RDF 문서를 SHACL 검증기에 입력하였다. 검증기가 검증 위반을 출력하지 않으면 통과로 판정하였다. 테스트에서 사용한 RDBMS는 MariaDB이고 SHACL 검증기는 dotNetRDF이다. 이 검증기는 2019년 W3C가 수행한 SHACL 검증기 테스트[22]에서 모든 항목을 통과하였다.

Table 6. Summary of Test Results

Class	RDF Test	SHACL Test	Class	RDF Test	SHACL Test
D000 (1)	P	P	D012 (1)	P	P
D001 (1)	P	P	D013 (1)	P	P
D002 (1)	P	P	D014 (1)	P	P
D003 (1)	P	P	D015 (1)	P	P
D004 (1)	P	P	D016 (1)	P	P
D005 (1)	P	P	D017 (1)	P	P
D006 (1)	P	P	D018 (1)	P	P
D007 (1)	P	P	D021 (1)	P	P
D008 (1)	P	P	D022 (1)	P	P
D009 (1)	P	P	D023 (1)	P	P
D010 (1)	P	P	D024 (1)	P	P
D011 (1)	P	P	D025 (1)	P	P

표 6은 테스트 결과를 요약한다. 표 6의 1, 4열은 테스트 범주명이며 괄호 안의 숫자는 각각의 범주에서 테스트에 사용된 테스트 케이스의 개수이며 결과적으로 문서 [21]이 DM을 위해 할당한 모든 테스트 케이스를 사용했

다. 표 6의 2, 5열은 RDF 테스트 결과이고 3, 6열은 SHACL 테스트 결과이다. 테스트 결과 열의 'P'는 통과(Pass)를 의미한다. 정리하면, 시스템은 문서 [21]이 정의한 24개의 DM 테스트 케이스에 대하여 DM을 준수하는 RDF 그래프를 생성했으며 생성된 그래프에 있는 모든 row node와 row node가 포함된 트리플 구조를 온전히 묘사할 수 있는 SHACL 스키마를 생성하였다.

그림 10~12는 범주 D009에 있는 DM을 위한 테스트 케이스의 RDB로부터 시스템이 출력한 RDF 그래프와 SHACL 스키마이며 세 가지 타입의 property shape가 모두 출력되는 사례 중 하나다. 그림 11은 sport 테이블에서 생성된 트리플에 대한 Shape들이며 그림 12는 student 테이블에서 생성된 트리플에 대한 shape들이다. 시스템은 shape의 종류마다 차이를 둔 IRI 작명 규칙을 적용해 사용자가 용이하게 의미를 파악할 수 있도록 하였다. node shape의 IRI는 그림 11~12의 1행처럼 원천 테이블명 뒤에 문자열 'Shape'가 결합된 형식이다. property shape의 IRI는 자신을 참조하는 node shape의 IRI 뒤에 Type1이면 '-col#', Type2이면 '-ref#', Type3이면 '-inverse#'의 접미사를 덧붙인 형식이다. 접미사의 마지막 문자 '#'는 자연수이며 생성 순서에 따라 1부터 오름차순으로 할당된다. 그림 11의 11, 19행과 그림 12의 12, 21, 29행이 Type1, 그림 12의 39행이 Type2, 그림 11의 29행이 Type3 property shape의 IRI다.

sport

ID (PK) INTEGER	Name VARCHAR(50)
100	Tennis

student

ID (PK) INTEGER	Sport (FK) INTEGER	Name VARCHAR(50)
10	100	Venus Williams
20	NULL	Demi Moore

```
01 <sport/ID=100>
02    a <sport> ;
03    <sport#ID> "100"^^xsd:integer ;
04    <sport#Name> "Tennis" .
05
06 <student/ID=10>
07    a <student> ;
08    <student#ID> "10"^^xsd:integer ;
09    <student#Name> "Venus Williams" ;
10    <student#Sport> "100"^^xsd:integer ;
11    <student#ref-Sport> <sport/ID=100> .
12
13 <student/ID=20>
14    a <student> ;
15    <student#ID> "20"^^xsd:integer ;
16    <student#Name> "Demi Moore" .
```

Fig. 10. The Test Case for Direct Mapping in D009

```
01 cse:sportShape
02    a sh:NodeShape ;
03    sh:targetClass d009:sport ;
04    sh:nodeKind sh:IRI ;
05    sh:class d009:sport ;
06    sh:pattern "^http://example.com/base/sport/ID=(.*)$" ;
07    sh:property cse:sportShape-col1 ;
08    sh:property cse:sportShape-col2 ;
09    sh:property cse:sportShape-inverse1  .
10
11 cse:sportShape-col1
12    a sh:PropertyShape ;
13    sh:path <http://example.com/base/sport#Name> ;
14    sh:nodeKind sh:Literal ;
15    sh:datatype xsd:string ;
16    sh:maxLength 50 ;
17    sh:maxCount 1  .
18
19 cse:sportShape-col2
20    a sh:PropertyShape ;
21    sh:path <http://example.com/base/sport#ID> ;
22    sh:nodeKind sh:Literal ;
23    sh:datatype xsd:integer ;
24    sh:minInclusive -2147483648 ;
25    sh:maxInclusive 2147483647 ;
26    sh:minCount 1 ;
27    sh:maxCount 1  .
28
29 cse:sportShape-inverse1
30    a sh:PropertyShape ;
31    sh:path [sh:inversePath
<http://example.com/base/student#ref-Sport> ] ;
32    sh:nodeKind sh:IRI ;
33    sh:class d009:student  .
```

Fig. 11. Shapes for triples from sport in Fig. 10

그림 11의 sportShape는 7~8행에서 2개의 Type1 property shape를 참조하며 그림 12의 studentShape는 7~9행에서 3개의 Type1 property shape를 참조한다. 이것은 sport 테이블이 2개 그리고 student 테이블이 3개의 컬럼으로 구성되었기 때문이다. 그림 11의 sportShape가 9행에서 참조하는 Type3 property shape와 그림 12의 studentShape가 10행에서 참조하는 Type2 property shape는 동일한 외래키 정의로부터 정의된 한 쌍의 property shape다. 이들은 그림 10의 11행 트리플을 묘사한다.

```
01 cse:studentShape
02    a sh:NodeShape ;
03    sh:targetClass d009:student ;
04    sh:nodeKind sh:IRI ;
05    sh:class d009:student ;
06    sh:pattern "^http://example.com/base/student/ID=(.*)$" ;
07    sh:property cse:studentShape-col1 ;
08    sh:property cse:studentShape-col2 ;
09    sh:property cse:studentShape-col3 ;
10    sh:property cse:studentShape-ref1  .
11
12 cse:studentShape-col1
13    a sh:PropertyShape ;
14    sh:path <http://example.com/base/student#Sport> ;
15    sh:nodeKind sh:Literal ;
16    sh:datatype xsd:integer ;
17    sh:minInclusive -2147483648 ;
18    sh:maxInclusive 2147483647 ;
19    sh:maxCount 1  .
20
21 cse:studentShape-col2
22    a sh:PropertyShape ;
23    sh:path <http://example.com/base/student#Name> ;
24    sh:nodeKind sh:Literal ;
25    sh:datatype xsd:string ;
26    sh:maxLength 50 ;
27    sh:maxCount 1  .
28
29 cse:studentShape-col3
30    a sh:PropertyShape ;
31    sh:path <http://example.com/base/student#ID> ;
32    sh:nodeKind sh:Literal ;
33    sh:datatype xsd:integer ;
34    sh:minInclusive -2147483648 ;
35    sh:maxInclusive 2147483647 ;
36    sh:minCount 1 ;
37    sh:maxCount 1  .
38
39 cse:studentShape-ref1
40    a sh:PropertyShape ;
41    sh:path <http://example.com/base/student#ref-Sport> ;
42    sh:nodeKind sh:IRI ;
43    sh:class d009:sport ;
44    sh:maxCount 1  .
```

Fig. 12. Shapes for triples from student in Fig. 10

그림 13은 범주 D016에 있는 DM을 위한 테스트 케이스에 있는 patient 테이블 생성을 위한 DDL 문과 그 테이블에 있는 한 행에 대응하는 시스템이 생성한 트리플들이다. 그림 14는 그림 13에 있는 9~12행의 트리플 묘사에 해당하는 property shape만을 포함한다.

```
01 CREATE TABLE patient (
02    ID INTEGER,
03    FirstName VARCHAR(50),
04    LastName VARCHAR(50),
05    Sex VARCHAR(6),
06    Weight REAL,
07    Height FLOAT,
08    BirthDate DATE,
09    EntranceDate TIMESTAMP,
10    PaidInAdvance BOOLEAN,
11    Photo VARBINARY(200),
12    PRIMARY KEY ("ID"));
```

```
01 <patient/ID=12>
02    a <patient> ;
03    <patient#ID> "12"^^xsd:integer ;
04    <patient#FirstName> "Chandler" ;
05    <patient#LastName> "Bing" ;
06    <patient#Sex> "male" ;
07    <patient#Weight> "90.31"^^xsd:double ;
08    <patient#Height> "1.76"^^xsd:double ;
09    <patient#BirthDate> "1978-04-06"^^xsd:date ;
10    <patient#EntranceDate> "2007-03-11T17:13:14Z"^^xsd:dateTime ;
11    <patient#PaidInAdvance> "true"^^xsd:boolean ;
12    <patient#Photo>
"89504E470D0A1A0A0000000D49484452000000050000000508060000
008D6F26E50000001C4944415408D763F9FFFEBFC37F062005C3201284
D031F18258CD04000EF535CBD18E0E1F0000000049454E44AE426082"^
^xsd:hexBinary .
```

Fig. 13. The Test Case for Direct Mapping in D016

그림 13~14는 시스템이 다양한 데이터 타입의 목적어를 갖는 literal triple에 대한 묘사가 가능함을 보이는 사례다. 그림 13의 9행, 10행은 각각 그림 14의 1~9행, 19~28행으로 묘사된다. xsd:date 혹은 xsd:dateTime 형식의 목적어가 준수해야 하는 RDF literal로서의 문자열 형식은 sh:pattern의 정규 표현식에 담겨있다. sh:minInclusive와 sh:maxInclusive의 값은 테스트에 사용된 MariaDB의 DATE와 TIMESTAMP 데이터 타입의 범위다. 특히, MariaDB에서의 TIMESTAMP는 타임존을 지원하는 데이터 타입이기 때문에 값의 범위 표현에 타임존이 표기되었다. 그림 13의 11행은 목적어의 타입이 xsd:boolean이다. 이 타입의 값의 범위는 그림 14의 16행에서 sh:in으로 한정되었다. RDBMS에 따라서 '0' 혹은 '1' 등 다른 표현 형식이 존재하기 때문에 이 제약조건으로 올바른 RDF 표현으로의 변환을 수행했는지를 검증할 수 있다. 마지막으로 그림 13의 12행 트리플은 그림 14의 30~36행으로 묘사된다. 이 트리플의 목적어는 이진 데이터의 RDF literal 표현이다. 그림 13에서 컬럼 Photo의 데이터 타입은 VARBINARY(200)이며 200은 이 컬럼의 최대 허용 바이트다. 이진 데이터는 xsd:hexBinary 표현으로 변환되며 이 데이터 타입은 1바이트를 2개의 16진수로 표현하기 때문에 sh:maxLengh의 값이 400으로 계산되었다.

```
01 cse:patientShape-col2
02    a sh:PropertyShape ;
03    sh:path <http://example.com/base/patient#BirthDate> ;
04    sh:nodeKind sh:Literal ;
05    sh:datatype xsd:date ;
06    sh:pattern "^\\d{4}-\\d{2}-\\d{2}$" ;
07    sh:minInclusive "1000-01-01"^^xsd:date ;
08    sh:maxInclusive "9999-12-31"^^xsd:date ;
09    sh:maxCount 1 .
10
11 cse:patientShape-col5
12    a sh:PropertyShape ;
13    sh:path <http://example.com/base/patient#PaidInAdvance> ;
14    sh:nodeKind sh:Literal ;
15    sh:datatype xsd:boolean ;
16    sh:in ( "true"^^xsd:boolean "false"^^xsd:boolean ) ;
17    sh:maxCount 1 .
18
19 cse:patientShape-col9
20    a sh:PropertyShape ;
21    sh:path <http://example.com/base/patient#EntranceDate> ;
22    sh:nodeKind sh:Literal ;
23    sh:datatype xsd:dateTime ;
24    sh:pattern "^\\d{4}-\\d{2}-
\\d{2}T\\d{2}:\\d{2}:\\d{2}(\\.{0}[\\d]{0,0}){0}Z$" ;
25    sh:minInclusive "1970-01-01 00:00:00Z"^^xsd:dateTime ;
26    sh:maxInclusive "2038-01-19 03:14:07Z"^^xsd:dateTime ;
27    sh:minCount 1 ;
28    sh:maxCount 1 .
29
30 cse:patientShape-col10
31    a sh:PropertyShape ;
32    sh:path <http://example.com/base/patient#Photo> ;
33    sh:nodeKind sh:Literal ;
34    sh:datatype xsd:hexBinary ;
35    sh:maxLength 400 ;
36    sh:maxCount 1 .
```

Fig. 14. Shapes for line 9~12 in Fig. 13

V. Conclusions

본 연구는 DM과 SHACL 이라는 W3C의 두 표준 사양에 기반한다. DM은 RDB로부터 RDF 그래프를 생성하기 위한 매핑규칙이며 SHACL은 RDF 그래프의 구조를 묘사하고 검증을 할 수 있는 RDF 그래프용 스키마 언어다. 본 연구에서는 DM으로 생성된 RDF 그래프에 대한 SHACL 스키마를 자동 생성하기 위한 방법을 제안했으며 이 방법을 구현한 시스템의 구조와 동작 과정을 기술함과 동시에 테스팅을 통해 제안한 방법의 유효성을 보였다. 본 연구의 결과가 기여코자 한 바는 두 가지다. 첫째, RDF 그래프의 구조 정보를 RDF로 표현 및 교환 가능하게 함으로써 SPARQL 사용자의 질의문 작성에 관한 생산성을 향상시키고자 한다. W3C는 SHACL을 “SHACL 문서이면 RDF 문서다.”라는 명제가 성립되도록 설계하였기 때문이다. 둘째, DM 기반으로 생성된 그래프가 가상 지식 그래프로서 서비스되는 조건에서 RDF 데이터가 외부로부터 입력될 때 RDBMS가 수행할 무결성 검사를 가상화 시스템이 자체적으로 수행하는 데 있어서 제안한 방법으로 생성한 SHACL 스키마가 그 용도로 사용되게끔 하는 것이다. 이를 위해 DM이 명시적으로 다루지 않는 RDB의 무결성 제약조건을 SHACL 스키마에 반영하였다. 후속 연구로서 SHACL 스키마를 기반으로 SPARQL 질의어의 자동 생성, 추천 생성, 유효성 검사를 수행하는 도구를 개발할 계획이다.

ACKNOWLEDGEMENT

All the code and test results of this work are hosted on github.com/jwchoi/Shape4RDB2RDF.

REFERENCES

[1] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," Semantic Web, Vol. 8, No. 3, pp. 489-508, Dec. 2017. DOI: 10.3233/SW-160218

[2] Q. Xu, X. Wang, J. Li, Q. Zhang, and L. Chai, "Distributed Subgraph Matching on Big Knowledge Graphs Using Pregel," IEEE Access, Vol. 7, pp. 116453-116464, August 2019. DOI: 10.1109/ACCESS.2019.2936465

[3] H. Arnaout and S. Elbassuoni, "Effective searching of RDF knowledge graphs," Journal of Web Semantics, Vol. 48, pp. 66-84, Jan. 2018. DOI: 10.1016/j.websem.2017.12.001

[4] J. Z. Pan, G. Vetere, J. M. Gomez-Perez, and H. Wu, "Exploiting Linked Data and Knowledge Graphs in Large Organisations," Springer, Cham, pp. 147-180, 2017.

[5] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao, “Semantic SPARQL similarity search over RDF knowledge graphs,” Proceedings of the VLDB Endowment, Vol. 9, No. 11, pp. 840-851, July 2016. DOI: 10.14778/2983200.2983201

[6] H. Knublauch and D. Kontokostas, "Shapes Constraint Language (SHACL)," https://www.w3.org/TR/shacl/

[7] H. Paulheim, "Machine Learning with and for Semantic Web Knowledge Graphs," Reasoning Web. Learning, Uncertainty, Streaming, and Scalability, pp 110-141, Esch-sur-Alzette, Luxembourg, September, 2018. DOI: 10.1007/978-3-030-00338-8_5.

[8] D. Buscaldi, D. Dess, E. Motta, F. Osborne, and D. R. Recupero, “Mining scholarly data for fine-grained knowledge graph construction,” Proceedings of the Workshop on Deep Learning for Knowledge Graphs Co-located with the 16th Extended Semantic Web Conference 2019, pp. 21-30, Portoroz, Slovenia, June, 2019.

[9] O. Corcho, F. Priyatna and D. Chaves-Fraga, "Towards a New Generation of Ontology Based Data Access," Semantic Web, Vol. 11, No. 1, pp. 153-160, January 2020. DOI: 10.3233/SW-190384

[10] M. Arenas, A. Bertails, E. Prud'hommeaux, and J. Sequeda, "A Direct Mapping of Relational Data to RDF," http://www.w3.org/TR/rdb-direct-mapping/

[11] S. Das, S. Sundara, and R. Cyganiak, "R2RML: RDB to RDF Mapping Language," http://www.w3.org/TR/r2rml/

[12] G. Xiao, L. Ding, B. Cogrel, and D. Calvanese, "Virtual Knowledge Graphs: An overview of systems and use cases," Data Intelligence, vol. 1, no. 3, pp. 201-223, May, 2019. DOI: 10.1162/dint_a_00011

[13] B. Motik, P. F. Patel-Schneider, and B. Parsia, "OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition)," http://www.w3.org/TR/owl-syntax/

[14] J. F. Sequeda, M. Arenas, and D. P. Miranker, "On directly mapping relational databases to RDF and OWL," Proceedings of the 21st international conference on World Wide Web, pp. 649-658, Lyon, France, April 2012. DOI: 10.1145/2187836.2187924

[15] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. Skjæveland, E. Thorstensen, and J. Mora, "BootOX: Practical Mapping of RDBs to OWL 2," The Semantic Web - ISWC 2015, pp. 113-132, Bethlehem, USA, October 2015. DOI: 10.1007/978-3-319-25010-6_7

[16] M. R. A. Rashid, G. Rizzo, M. Torchiano, N. Mihindukulasooriya, O. Corcho, and R. García-Castro, "Completeness and consistency analysis for evolving knowledge bases," Journal of Web Semantics, Vol 54, pp. 48-71, January 2019. DOI: 10.1016/j.websem.2018.11.004.

[17] M. O'Connor and A. Das, "SQWRL: a query language for OWL," Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529, pp. 208-215, Chantilly, USA, October 2009. DOI: 10.5555/2890046.2890072

[18] I. Kollia, B. Glimm, and I. Horrocks, "SPARQL Query Answering over OWL Ontologies," Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part I, pp. 382-396, Heraklion, Greece, May 2011. DOI: 10.1007/978-3-642-21034-1_26

[19] E. Sirin and B. Parsia, "SPARQL-DL: SPARQL Query for OWL-DL," Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions, Innsbruck, Austria, June 2007.

[20] D. Peterson, S. Gao, A. Malhotra, C. M. Sperberg-McQueen, and H. S. Thompson, "W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes," http://www.w3.org/TR/xmlschema11-2/

[21] B. Villazón-Terrazas and M. Hausenblas, "R2RML and Direct Mapping Test Cases," https://www.w3.org/TR/rdb2rdf-test-cases/

[22] J. E. L. Gayo, H. Knublauch and D. Kontokostas, "SHACL Test Suite and Implementation Report," https://w3c.github.io/data-shapes/data-shapes-test-suite/

Authors

Ji-Woong Choi received the B.S., M.S. and Ph.D. degrees in Computer Science and Engineering from Soongsil University, Korea, in 2001, 2003 and 2011, respectively. Dr. Choi joined the faculty of the School of Computer Science and Engineering at Soongsil University, Seoul, Korea, in 2013. He is currently an Associate Professor in the School of Computer Science and Engineering, Soongsil University. He is interested in information system.