Journal of KIIT. Vol. 17, No. 4, pp. 9-20, Apr. 30, 2019. pISSN 1598-8619, eISSN 2093-7571 9
http://dx.doi.org/10.14801/jkiit.2019.17.4.9

관계 DBMS에서 SQL 기반 그래프 패턴 질의 성능

박 우 창*

SQL Based Graph Pattern Query Performance on Relational DBMS

Uchang Park*

요 약

관계 데이터베이스에 저장된 데이터를 사용하는 그래프 응용의 경우 그래프 패턴질의 만을 위하여 따로 그래프 데이터베이스를 구축하기는 어려움이 있다. 본 연구는 이러한 어려움에 대하여 기존 관계 데이터베이스에서 SQL 언어를 이용한 그래프 패턴질의 처리 방법과 성능을 검토하였다. 첫째, 그래프 패턴 질의 유형에 대하여 SQL Server DBMS의 SQL Graph 질의와 오라클 DBMS의 SQL 질의를 이용한 해결 예를 보이고, 그래프 전용 DBMS의 Neo4j Cypher 질의 해결 예와 비교를 하였다. 둘째, 각각의 해결 예에 대하여 질의 처리 속도를 비교하고 성능 외에 질의 작성의 편의성, 성숙도, 유연성 등을 살펴보았다. 실험 결과 성능 면에서 SQL Server의 SQL Graph 질의는 노드나 에지가 바운드되지 않은 질의의 경우 Neo4j의 Cypher에 비하여 속도가 우수하며 그렇지 않은 경우는 Neo4j가 우수한 결과를 보였다. 성능 외적인 면에서는 관계 데이터베이스 백 엔드에서 그래프 질의 처리는 관계 데이터베이스의 유연성, 성숙된 기능을 사용할 수 있는 장점이 있다.

Abstract

It is very difficult to build a separate graph database application for graph pattern query that uses data stored in relational DBMS. To overcome these difficulties, in this paper, we discuss graph pattern query processing methods and performance using SQL on relational DBMS. First, for the graph pattern queries, we show example solutions for SQL Graph query of SQL Server and SQL query of Oracle DBMS, and also for Cypher of Neo4j graph oriented DBMS. Second, we compare query processing speed of each example solutions and also compare for ease of query programming, maturity and flexibility. In the experiment, we find that SQL Graph of SQL Server is more better in query processing speed than Cypher of Neo4j for those queries that nodes and edges are not bounded, but Cypher of Neo4j is better in the other cases. Other than the query processing speed, SQL graph query has advantages in utilizing maturities and supports of relational database.

Keywords
relational database, graph database, graph pattern query, SQL graph, SQL, cypher

* 덕성여자대학교 컴퓨터학과 교수
- ORCID: http://orcid.org/0000-0002-7691-3944

· Received: Dec. 31, 2018, Revised: Feb. 28, 2019, Accepted: Mar. 03, 2019
· Corresponding Author: Uchang Park
Dept. of Computer Science, Duksung Women's University, Seoul, Korea
Tel.: +82-2-901-8342, Email: ucpark@duksung.ac.kr

Ⅰ. 서 론

그래프 데이터베이스는 패턴의 탐색, 추천 시스템 등 응용에 많이 사용되고 있으며, 대용량 그래프 응용의 경우 그래프 데이터베이스는 관계 데이터베이스보다 더 성능이 좋은 것으로 알려져 있다[1]. 그러나 일반적인 응용에서 데이터 발생 및 수집은 관계 데이터베이스로 구축되기 때문에 그래프 패턴 질의가 필요한 경우 관계 데이터베이스에서 그래프 질의를 해야 하는 경우가 많다[2]. 따라서 그래프의 분석질의 혹은 패턴질의만을 위하여 그래프 데이터베이스를 따로 구축하기는 어려움이 있다[3].

그래프 질의는 분석질의와 패턴질의로 나눌 수 있는데 최단거리 문제 같은 분석질의는 그래프 데이터베이스의 성능이 뛰어나고 패턴질의의 경우는 데이터베이스 시스템에 따라 다른 성능을 보이는 것으로 알려져 있다[2]. 그래프 패턴 질의는 그래프 데이터베이스에서 중요한 질의이며 관계 데이터베이스 질의로 변환해서 수행해야할 경우 질의 형태가 조인 문으로 수행되기 때문에 문장이 복잡하게 나타난다[4].

그래프 데이터베이스의 대표적인 시스템은 Neo4j [5] 전용 엔진을 사용한다. 질의 언어는 Cypher를 사용하며 Property Graph에 대한 강력한 질의 기능을 갖고 있다. 2016년 openCypher 프로젝트를 통하여 그래프 질의 언어 표준화를 시작하였다.

최근 상용 관계 데이터베이스 엔진에도 그래프 질의 구문을 추가하여 분석질의나 패턴 질의 성능 향상을 도모하고 있다.

오라클(Oracle) DBMS는 12c 버전에 그래프 질의를 수행하기 위한 방법으로 오라클의 Property Graph 개념을 사용하여 메모리 기반 그래프 데이터베이스를 제공한다. 질의는 SQL 언어에 그래프 분석 함수를 추가하여 그래프 질의를 제공한다. 인터페이스로는 PL/SQL, Java, PGQL, Python 같은 언어를 사용한다[6].

마이크로소프트는 SQL Server 2017 버전에 그래프 질의를 수행하기 위한 방법으로 SQL 언어를 확장하여 SQL 언어 WHERE 절에 Cypher 언어 유사 문법을 추가하여 그래프 데이터베이스와 연결을 시도하였다(앞으로 SQL Graph로 호칭하기로 한다). 관계 데이터베이스 엔진에서 그래프와 관계 데이터를 같이 처리하는 방법으로 관계 데이터베이스의 성숙한 도구들을 같이 사용할 수 있는 장점이 있다. [7].

본 연구는 그래프 데이터베이스를 별도로 구축하지 않고 기존 관계 데이터베이스에서 그래프 패턴 질의 처리를 할 경우 SQL 질의 작성의 용이성과 질의 처리 속도 비교를 통하여 관계 및 그래프 DBMS의 상호 유용성을 비교하고자 한다. 비교 대상 질의 언어는 그래프 데이터베이스 Neo4j의 Cypher, 관계 데이터베이스 SQL Server의 SQL Graph, 오라클의 SQL 3가지에 대하여 비교한다. 먼저 샘플 데이터베이스를 구축하여 질의 작성 예를 보이고 작성된 질의의 성능을 비교하며, 그 외 질의의 편의성 등도 같이 비교해보고자 한다.

실험 데이터는 시맨틱웹 데이터베이스 벤치마킹에서 사용되며 많은 그래프 데이터베이스 그래프 벤치마킹에 사용되는 LUBM 데이터를 사용한다 [8]-[10].

2장에서는 비교를 위한 데이터 준비 및 그래프 패턴 질의에 대한 설명을 하고, 3장에서는 성능 실험 및 결과, 4장에서는 질의 언어의 질적인 비교를 하고, 5장에서는 결론 및 향후 연구에 대하여 논한다.

Ⅱ. 관련 연구

2.1 연구 동향

그래프 응용에서 순수 그래프 데이터베이스는 성능이 우수하며, 관계형 데이터베이스에서 그래프 데이터 처리는 트랜잭션, 신뢰성 있는 접근제어에 장점이 있다.

순수 그래프 데이터베이스의 대표 제품은 Neo4j로 그래프 데이터베이스 시장의 46%를 점유하고 있다. 관계 데이터베이스에 그래프 데이터를 처리하기 위하여 그래프 질의 구문을 추가한 제품은 SQL Server 2017 버전의 SQL Graph이다[7]. SQL Graph는 관계 데이터베이스와 같은 테이블 구조를 갖고 노드와 에지를 SQL CREATE 문으로 선언을 하고 SQL의 WHERE 절에서 MATCH 문을 통하여 그래프 질의를 한다.

그래프 응용에 대한 Cypher 언어의 그래프 데이터베이스와 SQL의 관계 데이터베이스의 처리 성능 비교는 많은 연구를 통하여 이루어졌다.

Chad Vicknair[1]은 Neo4j와 MySQL 데이터베이스에서 노드와 에지로 구성된 단순 DAG(Directed Acyclic Graph)를 노드수를 4가지(1000, 5000, 10000, 100000)로 변화시키면서 경로질의의 수행 속도와 질의의 용이성, 확장성 등을 비교하였다. 수행 속도는 노드 수가 작은 경우를 제외하고는 대부분 Neo4j가 수행 속도 면에서 MySQL보다 우수하였다.

Adam Welc[11]는 Neo4j와 오라클 데이터베이스에서 노드와 에지로 구성된 단순 DAG(Directed Acyclic Graph)에 저장된 소셜 그래프의 최단거리 문제의 예를 들어 그래프 질의에서 Neo4j가 SQL에 비하여 우수하지 않을 수도 있다는 것을 보였다.

Andrey Gubichev[9]는 Neo4j와 Virtuoso 관계 데이터베이스에서 LUBM 데이터를 이용하여 SPARQL 그래프 패턴매칭 질의에 대한 성능을 측정하였다.

Jürgen Hölsch[10]는 Neo4j와 상용 관계 데이터베이스에서 LUBM 벤치마크를 이용하여 그래프 분석 질의와 패턴질의에 대하여 Cypher와 SQL 질의의 수행 속도와 질의의 용이성, 확장성 등을 비교하였다. 질의 수행 속도는 분석질의는 SQL이 Cypher 보다 우수하고, 패턴질의의 경우는 패턴 경로의 길이가 작으면 SQL이, 길이가 커지면 Neo4j의 Cypher 질의가 우수함을 보였다. 그 외, 그래프 데이터베이스에 대한 많은 연구가 다양한 DBMS와 질의 타입에 대하여 비교 연구되어 왔다[12][13].

기존의 연구는 대표적인 그래프 데이터베이스인 Neo4j와 MySQL, 오라클 등 관계 데이터베이스와의 성능 비교가 주로 이루어졌으며 그래프 질의의 유형에 따라 성능 평가 결과가 조금씩 다르게 나타난다. 본 연구는 Jürgen Hölsch의 세분화된 그래프 패턴 질의에 대하여 기존의 관계 데이터베이스 SQL 언어를 확장한 SQL Server DBMS의 SQL Graph를 중심으로 그래프 패턴 질의 작성의 용이성과 성능 평가를 시도하였다.

2.2 실험 데이터베이스 및 벤치마크 데이터

기존의 연구는 그래프 질의를 그래프 데이터베이스와 관계 데이터베이스에서 처리를 하고 성능 및 질적인 비교를 수행하였다. 본 연구는 관계 데이터베이스의 SQL 질의어를 확장하여 관계 데이터베이스 엔진에서 그래프 데이터베이스의 Cypher 언어를 처리하는 SQL Graph와 비교를 한다. 추가적으로 관계 데이터베이스의 대표인 오라클 데이터베이스의 SQL 언어에 대한 비교를 포함한다. 연구의 개요도는 그림 1과 같다.

그래프 데이터베이스 성능 측정을 위한 표준화된 벤치마크 데이터는 따로 없지만 시맨틱 웹에서 사용하는 RDF 그래프 모델의 LUBM 벤치마크 데이터를 사용한다. LUBM은 Lehigh 대학에서 개발한 것으로 도메인이 대학 운영인 데이터이다[14]. LUBM 벤치마크 데이터는 OWL 포맷으로 생성된다. OWL 형식 데이터는 그래프 데이터에 적재하기 위하여 변환기를 이용하여 GraphML로 바꾼다.

3가지 데이터베이스의 스키마 설계와 DML 생성 과정은 그림 2와 같다.

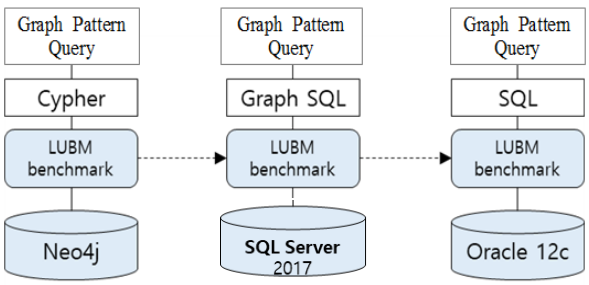

그림 1. 데이터베이스 질의 비교 플랫폼
Fig. 1. Database query comparison platform

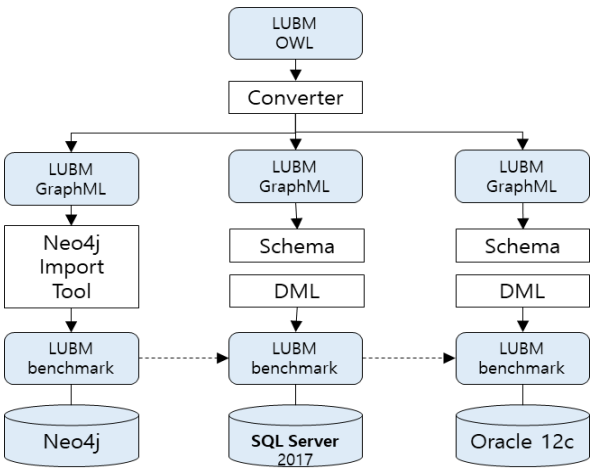

그림 2. 벤치마크 데이터 변환 과정
Fig. 2. Benchmark data conversion

(1) Neo4j 데이터 생성

LUBM 벤치마크 데이터는 GraphML 데이터 형식으로 바꾸어 Neo4j의 Import 도구를 이용하여 적재한다. 적재 과정에서 그래프 데이터의 노드, 에지 타입, 데이터가 표 1과 같이 생성된다. 적재된 노드 타입은 12개, 노드 수는 38,334개 에지 타입은 12개, 에지 수는 113,464개이다.

표 1. 그래프 데이터 내역
Table 1. Graph data contents

Node Type	n	Edge type starting from node(number of edges)
University	1,000	
Department	34	subOrganizationOf (552)
Full Professor	289	doctoralDegreeFrom (1,242) mastersDegreeFrom (1,242) undergraduateDegreeFrom (5,561) worksFor (1,242) teacherOf (3,728) headOf (34) publicationAuthor (24,688)
Associate Professor	410	
Assistant Professor	333	
Lecturer	210	
Undergraduate Student	13,559	memberOf (17,878) takesCourse (49,305)
Graduate Student	4319	advisor (7,044) teachingAssistantOf (947)
Course	1,889	
GraduateCourse	1,839	
ResearchGroup	518	
Publication	13934	
Total	38,334	113,464

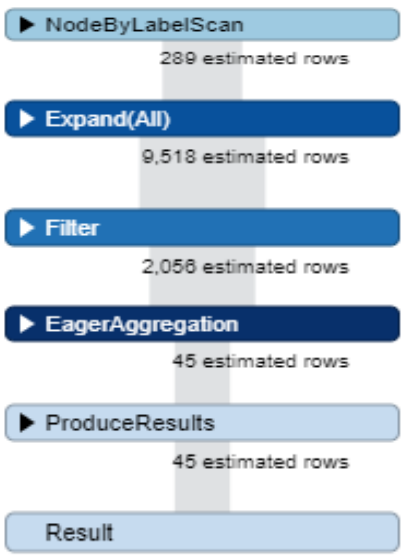

그림 3. Neo4j 실행계획 예
Fig. 3. Neo4j example execution plan

Neo4j의 Cypher는 MATCH 문에서 그래프 경로에 따라 질의를 하며 결과는 RETURN 문을 통하여 반환된다. 예를 들어 교수의 강좌를 찾는 경우 질의는 다음과 같다. Neo4j 그래프 DBMS 질의의 실행 계획은 NodeByLabelScan, Expand, Filter 등의 연산을 사용한다.

```
MATCH (a:FullProfessor)--(b:Course)
RETURN count(*)
```

(2) SQL Server용 데이터 생성

LUBM GraphML 형식의 데이터로 부터 SQL Server의 SQL Graph 데이터 스키마 설계는 수작업으로 진행하였으며, 노드는 노드 테이블로, 에지는 에지 테이블로 스키마 설계를 한다.

스키마 생성 과정에서 그래프 노드는 노드 테이블로, 에지는 에지 테이블로 변환한다.

노드 테이블 생성의 예를 들면 다음과 같다.

```
CREATE TABLE departments (
  departmentid smallint NOT NULL,
  universityid smallint NOT NULL) AS NODE;
```

에지 테이블 생성의 예로 교수와 학과의 소속 관계를 나타내는 worksFor는 다음과 같은 에지로 생성된다.

```
CREATE TABLE worksFor AS EDGE;
```

그래프 노드를 관계 테이블로 변환하는 방법은 유사한 노드 타입을 통합된 테이블로 구성하는 방법과 그래프 노드를 1:1로 테이블로 변환하는 방법이 있다. 이 경우 장단점이 있는데 본 연구에서는 관계 데이터베이스 설계 시와 가까운 전자의 방법을 선택하였다[15]. 예를 들면 교수를 나타내는 FullProfessor, AssociateProfessor, AssistantProfessor, Lecturer 4가지 노드는 관계 데이터베이스에서는 Teacher 테이블로 통합한다. Teacher 테이블에 ttype 속성을 두어 교수의 타입을 구분한다.

GraphML 데이터로부터 DML 생성은 본 논문에서 Python 프로그램으로 제작하였다.

그림 4는 SQL Graph 용으로 설계된 스키마 구조를 나타낸다.

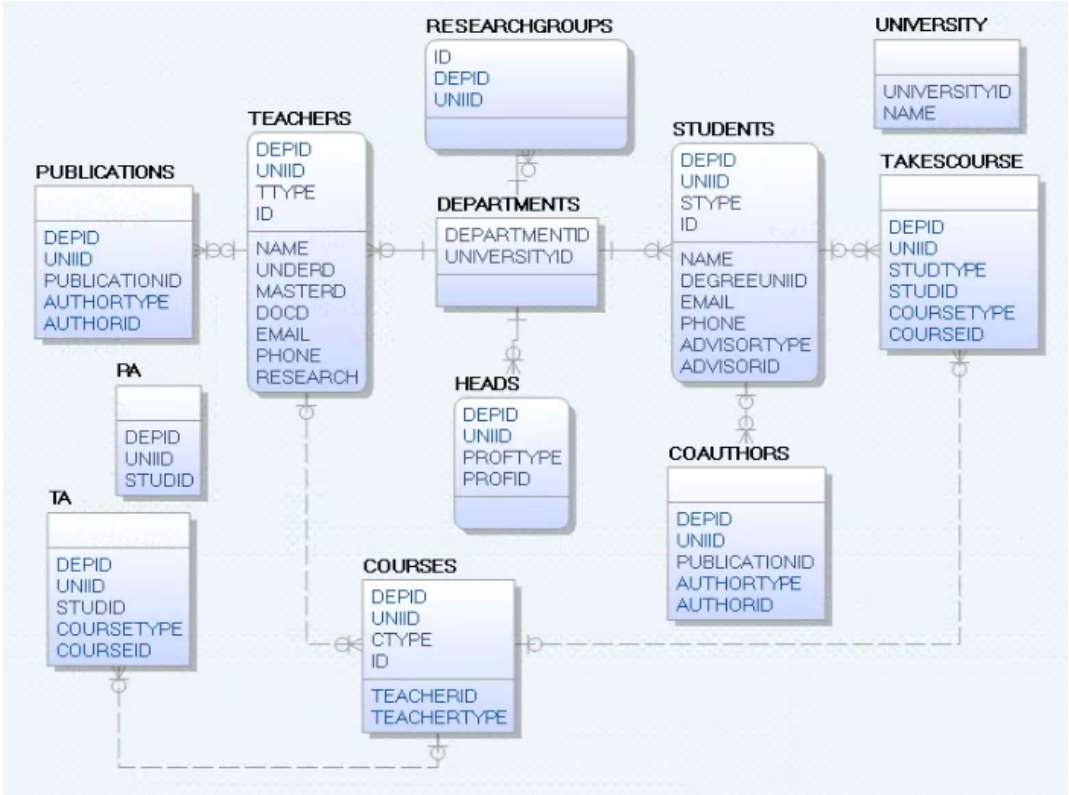

그림 4. SQL Graph용 스키마
Fig. 4. Schema for SQL graph

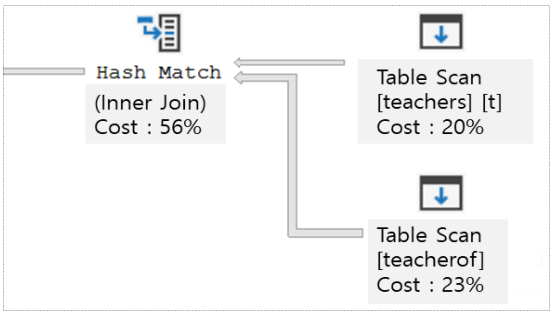

그림 5. SQL Graph의 실행 계획 예
Fig. 5. Example execution plan of SQL graph

SQL Server의 SQL Graph는 SQL 구문에 MATCH 문을 추가하여 사용한다. MATCH 문은 Neo4j의 MATCH 문과는 달리 에지에 반드시 바운드되는 에지 이름을 적도록 제한되어 있다. 예를 들어 교수의 강좌를 찾는 경우 질의는 다음과 같다. SQL Server의 실행 계획은 Hash Match 연산을 사용한다.

```
SELECT count(*)
FROM  teachers t, teacherof, courses c
WHERE MATCH(t-(teacherof)->c)
   AND t.ttype=3 AND c.ctype=0;
```

(3) 오라클용 데이터 생성

LUBM 벤치마크 데이터의 관계 데이터베이스 변환은 먼저 스키마 생성 그리고 DML 생성이 필요하다. 오라클의 관계 데이터 스키마 설계는 노드와 다대다의 관계는 테이블로, 일대 다 관계 에지는 외래키로 설계를 하였다. 노드의 설계는 (2)의 방법과 같은 유사한 노드 타입을 통합된 테이블로 구성하는 방법을 사용하였다. 그래프 에지는 외래키 관계로 저장한다. 예를 들어 worksFor 관계는 Teacher에 Department의 기본키를 외래키로 저장하여 나타낸다. GraphML 데이터로 부터 DML 생성은 마찬가지로 본 논문에서 Python 프로그램으로 제작하였다.

2.3 그래프 패턴 질의

그래프 질의는 패턴 질의(Pattern Matching Query)와 분석 질의(Analytical Query)로 나눈다[10]. 패턴질의는 노드 레이블로 경로 필터링, 에지 레이블로 경로 필터링, 싸이클 등 그래프의 일부분을 접근하는

질의이고, 분석질의는 노드 차수, 차수의 집중도, 연결성, 최단 거리 계산 등 그래프의 많은 부분을 접근하는 질의이다.

(정의) 그래프 패턴 P

그래프 G=(V, E)가 주어졌을 때, 그래프 패턴 P는 P=(Vp, Ep)로 나타내며 부분 그래프가 노드와 에지에 대한 조건 p를 각각 만족한다.

본 연구에서는 그래프 패턴 질의의 대표적인 타입 4가지에 대하여 질의를 Cypher(Neo4j)와 SQL Graph(SQL Server), SQL(Oracle) 언어로 실험을 한다. 4가지 타입은 Jürgen Hölsch[10] 연구에서 구분한 방법으로, 노드 레이블로 경로 필터링, 에지 레이블로 경로 필터링, 노드/에지 레이블로 경로 필터링, 사이클 경로 타입의 질의이다. 성능 평가를 좀 더 정밀하게 하기 위하여 4가지 질의 패턴 타입 각각에 단순질의, 복잡질의의 2가지 형으로 나누어 실험한다.

질의의 타입과 실험을 위한 LUBM 벤치마크 예제 질의 4가지 타입, 8가지 질의는 다음과 같다.

(1) 패턴 질의 타입 1: 노드 레이블로 경로 필터링 : 그래프 노드 레이블로 필터링하는 질의로 에지에는 제한을 두지 않는 경우이다. 즉 탐색 에지를 질의에 바운드 시키지 않은 경우이다. SQL Graph나 SQL은 에지에 해당되는 교차테이블이나 외래키를 반드시 바운드 시켜야 한다.

(Q 1: 단순형) 정교수(FullProfessor)의 강좌를 찾아내는 질의를 구하여라.

(Cypher)

```
MATCH (a:FullProfessor)--(b:Course)
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM  teachers t, teacherof, courses c
WHERE MATCH(t-(teacherof)->c)
    AND t.ttype=3 AND c.ctype=0;
```

(SQL)

```
SELECT count(*)
FROM teachers t, courses c
WHERE t.uniid=c.uniid AND t.depid=c.depid
    AND t.id=c.teacherid AND t.ttype=c.teachertype
    AND t.ttype=3 c.ctype=0;
```

SQL 문의 경우 복합키를 사용하였기 때문에 질의가 길어지는 불가피함이 발생한다. 예를 들어 Teacher 테이블의 키는 (대학번호, 학과번호, 교수번호, 교수타입)으로 (uniid, deptid, teacherid, ttype)가 된다. 스키마 생성시 인조키를 만들어 기본키로 설정했다면 질의가 다음과 같이 간단해진다.

```
SELECT count(*)
FROM teachers t, courses c
WHERE t.id=c.teacherid;
```

그러나 LUBM 벤치마크에 노드에 기본키가 없으므로 본 연구에서는 테이블 변환시 복합키를 그대로 사용하였다.

(Q 2: 복잡형) 정교수(FullProfessor)의 학과, 학과소속학생, 학생의 수강강좌를 찾아내는 질의를 구하여라.

(Cypher)

```
MATCH (b:FullProfessor)--(c:Department)
        --(d:UndergraduateStudent)--(e:Course)
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM teachers t, worksfor, departments d,
        memberof, students s, takescourse, courses c
WHERE
    MATCH (t-(worksfor)->d<-(memberof)-s
                -(takescourse)->c)
    AND t.ttype=3 AND s.stype=0 AND c.ctype=0;
```

(SQL)

```
SELECT count(*)
FROM university u, teachers t, departments d,
        students s, takescourse tc, courses c
WHERE u.universityid=t.uniid
    AND t.uniid=d.universityid
    AND t.depid=d.departmentid
    AND d.universityid=s.uniid
    AND d.departmentid=s.depid
```

```
    AND s.uniid=tc.uniid AND s.depid=tc.depid
    AND s.stype=tc.studtype AND s.id=tc.studid
    AND tc.uniid=c.uniid AND tc.depid=c.depid
    AND tc.coursetype=c.ctype AND tc.courseid=c.id
    AND t.ttype=3 AND s.stype=0 AND c.ctype=0;
```

(2) 패턴 질의 타입 2: 에지 레이블로 경로 필터링 : 그래프 에지 레이블로 필터링하는 질의로 노드에는 제한을 두지 않는 경우이다. 즉 탐색 노드를 질의에 바운드 시키지 않은 경우이다. SQL Graph나 SQL은 노드에 해당되는 테이블을 반드시 바운드 시켜야 한다.

(Q 3: 단순형) 교수의 강좌와 강좌를 듣는 수강학생들을 보여라.

(Cypher)

```
MATCH (a)-[:teacherOf]->(b)<-[:takesCourse]-(c)
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM teachers t, teacherof, courses c,
        takescourse, students s
WHERE MATCH (t-(teacherOf)->c<-(takescourse)-s);
```

(SQL)

```
SELECT count(*)
FROM teachers t, courses c,
        takescourse tc, students s
WHERE t.uniid=c.uniid AND t.depid=c.depid
    AND t.id=c.teacherid AND t.ttype=c.teachertype
    AND c.uniid=tc.uniid AND c.depid=tc.depid
    AND c.id=tc.courseid AND c.ctype=tc.coursetype
    AND tc.uniid=s.uniid AND tc.depid=s.depid
    AND tc.studid=s.id AND tc.studtype=s.stype;
```

(Q 4: 복잡형) 교수의 강좌를 듣는 학생의 지도교수의 석사학위 대학, 같은 대학 박사학위 교수 관계를 보여라.

(Cypher)

```
MATCH (a)-[:teacherOf]->(b)<-[:takesCourse]-(c)
        -[:advisor]->(d)-[:mastersDegreeFrom]->(e)
        <-[:doctoralDegreeFrom]-(f)
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM teachers t, teacherof, courses c, takescourse,
        students s, advisor, teachers tt,
        mastersDegreeFrom, university u,
        doctoralDegreeFrom, teachers ttt
WHERE
    MATCH (t-(teacherof)->c<-(takesCourse)-s
        -(advisor)->tt-(mastersDegreeFrom)->u
        <-(doctoralDegreeFrom)-ttt);
```

(SQL)

```
SELECT count(*)
FROM teachers t, courses c, takescourse tc,
        students s, teachers tt, university u,
        teachers ttt
WHERE t.uniid=c.uniid AND t.depid=c.depid
    AND t.ttype=c.teachertype AND t.id=c.teacherid
    AND c.uniid=tc.uniid AND c.depid=tc.depid
    AND c.ctype=tc.coursetype AND c.id=tc.courseid
    AND tc.uniid=s.uniid AND tc.depid=s.depid
    AND tc.studtype=s.stype AND tc.studid=s.id
    AND s.uniid=tt.uniid AND s.depid=tt.depid
    AND s.advisortype=tt.ttype AND s.advisorid=tt.id
    AND tt.masterD=u.universityid
    AND ttt.docD=u.universityid;
```

(3) 패턴 질의 타입 3: 노드/에지 레이블로 경로 필터링 : 그래프 노드 및 에지 레이블로 필터링하는 질의이다.

(Q 5: 단순형) 정교수의 강좌와 강좌를 듣는 학부학생들을 보여라.

(Cypher)

```
MATCH (a:FullProfessor)-[:teacherOf]->(b:Course)
        <-[:takesCourse]-(c:UndergraduateStudent)
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM teachers t, teacherof, courses c,
        takescourse, students s
WHERE MATCH (t-(teacherOf)->c<-(takescourse)-s)
    AND t.ttype=3 AND c.ctype=0 AND s.stype=0;
```

(SQL)

```
SELECT count(*)
FROM teachers t, courses c,
        takescourse tc, students s
```

```
WHERE t.uniid=c.uniid AND t.depid=c.depid
    AND t.id=c.teacherid AND t.ttype=c.teachertype
    AND c.uniid=tc.uniid AND c.depid=tc.depid
    AND c.id=tc.courseid AND c.ctype=tc.coursetype
    AND tc.uniid=s.uniid AND tc.depid=s.depid
    AND tc.studid=s.id AND tc.studtype=s.stype
    AND t.ttype=3 AND c.ctype=0 AND s.stype=0;
```

(Q 6: 복잡형) 정교수의 강좌와 강좌를 듣는 학부학생들의 부교수인 지도교수의 석사학위 대학을 보여라.

(Cypher)

```
MATCH (a:FullProfessor)-[:teacherOf]->(b:Course)
        <-[:takesCourse]-(c:UndergraduateStudent)
        -[:advisor]->(d:AssociateProfessor)
        -[:mastersDegreeFrom]->(e:University)
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM teachers t, teacherof, courses c, takescourse,
        students s, advisor, teachers tt,
        mastersDegreeFrom, university u
WHERE
    MATCH (t-(teacherOf)->c<-(takescourse)-s-
        (advisor)->tt-(mastersDegreeFrom)->u)
    AND t.ttype=3 AND c.ctype=0  AND s.stype=0
    AND tt.ttype=2;
```

(SQL)

```
SELECT count(*)
FROM teachers t, courses c,  takescourse tc,
        students s, teachers tt, university u
WHERE t.uniid=c.uniid AND t.depid=c.depid
    AND t.ttype=c.teachertype AND t.id=c.teacherid
    AND c.uniid=tc.uniid AND c.depid=tc.depid
    AND c.ctype=tc.coursetype AND c.id=tc.courseid
    AND tc.uniid=s.uniid AND tc.depid=s.depid
    AND tc.studtype=s.stype AND tc.studid=s.id
    AND s.uniid=tt.uniid AND s.depid=tt.depid
    AND s.advisortype=tt.ttype AND s.advisorid=tt.id
    AND tt.masterD=u.universityid
    AND t.ttype=3 AND c.ctype=0 AND s.stype=0
    AND tt.ttype=2;
```

(4) 패턴 질의 타입 4: 노드의 사이클

(Q 7: 단순형) 학부학생의 지도교수 중 정교수인 교수와 정교수의 소속학과 관계를 찾아라.

(Cypher)

```
MATCH (a:UndergraduateStudent)--(b:FullProfessor)
        --(c:Department)--(a)
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM students s, advisor, teachers t,
        worksfor, departments d, memberof
WHERE MATCH(s-(advisor)->t-(worksfor)->d
        <-(memberof)-s)
    AND s.stype=0 AND t.ttype=3;
```

(SQL)

```
SELECT count(*)
FROM students s, teachers t, departments d
WHERE s.uniid=t.uniid AND s.depid=t.depid
    AND s.advisorid=t.id AND s.advisortype=t.ttype
    AND t.uniid=d.universityid
    AND t.depid=d.departmentid
    AND d.universityid=s.uniid
    AND d.departmentid=s.depid
    AND t.ttype=3 AND s.stype=0;
```

(Q 8: 복잡형) 학부학생의 수강과목의 교수가 부교수인 경우에 대한 지도학생의 수강강좌, 수강강좌의 강사가 조교수이며 처음학생의 지도교수를 찾아라.

(Cypher)

```
MATCH (a:UndergraduateStudent)--(b:Course)
        --(c:AssociateProfessor)
        --(d:UndergraduateStudent)
        --(e:Course) --(f:AssistantProfessor)--(a)
WHERE b <> e AND a <> d
RETURN count(*)
```

(SQL Graph)

```
SELECT count(*)
FROM students s, takescourse tc, courses c,
        teacherof to1, teachers t, advisor a,
        students ss, takescourse tcc, courses cc,
        teacherof to2, teachers tt, advisor aa
WHERE
```

```
    MATCH (s-(tc)->c<-(to1)-t
            <-(a)-ss-(tcc)->cc<-(to2)-tt<-(aa)-s)
    AND s.stype=0 AND c.ctype=0
    AND t.ttype=2 AND ss.stype=0
    AND cc.ctype=0 AND tt.ttype=1
    AND NOT (c.uniid=cc.uniid
          AND c.depid=cc.depid
          AND c.ctype=cc.ctype AND c.id=cc.id)
    AND NOT (s.uniid=ss.uniid
           AND s.depid=ss.depid
           AND s.stype=ss.stype AND s.id=ss.id);
(SQL)
SELECT count(*)
FROM students s, takescourse tc,courses c,
           teachers t, students ss, takescourse tcc,
           courses cc, teachers tt
WHERE s.uniid=tc.uniid AND s.depid=tc.depid
    AND s.stype=tc.studtype AND s.id=tc.studid
    AND tc.uniid=c.uniid AND tc.depid=c.depid
    AND tc.coursetype=c.ctype AND tc.courseid=c.id
    AND c.uniid=t.uniid AND c.depid=t.depid
    AND c.teachertype=t.ttype AND c.teacherid=t.id
    AND t.uniid=ss.uniid AND t.depid=ss.depid
    AND t.ttype=ss.advisortype AND t.id=ss.advisorid
    AND ss.uniid=tcc.uniid AND ss.depid=tcc.depid
    AND ss.stype=tcc.studtype AND ss.id=tcc.studid
    AND tcc.uniid=cc.uniid AND tcc.depid=cc.depid
    AND tcc.coursetype=cc.ctype AND tcc.courseid=cc.id
    AND cc.uniid=tt.uniid AND cc.depid=tt.depid
    AND cc.teachertype=tt.ttype AND cc.teacherid=tt.id
    AND tt.uniid=s.uniid AND tt.depid=s.depid
    AND tt.ttype=s.advisortype AND tt.id=s.advisorid
    AND NOT ((c.uniid = cc.uniid)
    AND (c.depid = cc.depid)
    AND (c.teachertype = cc.teachertype)
    AND (c.teacherid = cc.teacherid))
      AND NOT ((s.uniid = ss.uniid)
    AND (s.depid = ss.depid)
    AND (s.stype = ss.stype) AND (s.id = ss.id))
    AND s.stype=0 AND c.ctype=0
    AND t.ttype=2 AND ss.stype=0
    AND cc.ctype=0 AND tt.ttype=1;
```

III. 실험 및 결과

3.1 실험 환경

실험을 위한 컴퓨터는 Windows 10 Home, Intel Premium CPU G4400 3.30GHz, 8GB RAM을 사용하였다. 그래프 데이터베이스는 neo4j 3.4.1, 관계형 데이터베이스는 오라클 12c, 그래프 SQL은 SQL Server 2017을 사용하였다. LUBM 그래프 데이터를 SQL 데이터로 변환하는 DML 생성은 Python 프로그램으로 작성하였다.

패턴질의는 앞 절에서 설명한 4가지 패턴의 단순형/복잡형 8가지 질의(Q1~Q8)를 테스트하였다. DBMS 결과를 출력하는 오버헤드가 다르기 때문에 패턴질의의 결과는 count(*)를 출력하도록 하여 IO 지연 시간을 제거하였다. 수행시간은 8가지 질의 타입에 대하여 각각 5번 무작위 실험하여 최대/최소를 제거한 3개 평균값으로 하였다. 실험 결과 각 질의마다 매 실험 중 첫 번째 실험이 시간이 많이 걸린다. 3가지 DBMS 모두 비슷한 현상이지만 특히 오라클은 첫 번째 질의 시간이 가장 많이 걸리고 이후 수행시간이 최적화 작동으로 인하여 급속히 감소한다. 실험에서는 두 번째 이후 시간을 측정하였다.

질의의 실행계획을 살펴보면 Neo4j의 Cypher는 All Node Scan(Q3, Q4) 혹은 Node by Label Scan (Q1-Q2, Q5-Q8)이었으며, SQL Server의 SQL Graph는 Hash Match, 오라클의 SQL은 Hash Join으로 실행되었다.

3.2 실험 결과 비교

각 질의들의 실행 시간 평균은 표 2와 같다.

표 2의 실험 결과를 보면 다음을 관찰할 수 있었다.

(1) MATCH 문을 사용하는 Neo4j의 Cypher와 SQL Server의 SQL Graph를 비교해보면 Cypher는 SQL Graph에 비하여 그래프의 노드 혹은 에지가 바운드되지 않은 패턴 타입 1과 패턴 타입 2와 같은 경우 수행시간이 더 크다(Q1~Q4). 그러나 패턴

타입 3과 패턴 타입 4와 같이 탐색해야할 에지나 노드가 바운드되어 정해진 질의의 경우 SQL Graph 보다 수행시간이 더 작다(Q5~Q8). 그림 6은 Neo4j Cypher와 SQL Graph를 비교한 것이다.

표 2. 데이터베이스 별 질의의 수행 시간(ms)
Table 2. Query execution time for each database(ms)

Query Type	style	Query name	Neo4j Cypher	SQL Graph	Oracle SQL
1	simple	Q1	4	1	1
	complex	Q2	176	106	23
2	simple	Q3	40	17	12
	complex	Q4	78	23	150
3	simple	Q5	7	23	10
	complex	Q6	12	22	93
4	simple	Q7	9	9	3
	complex	Q8	23	61	114

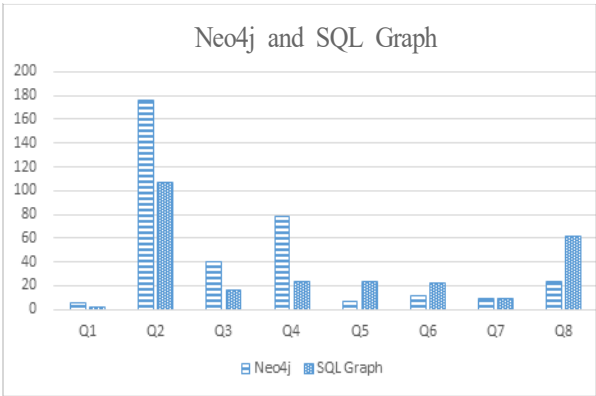

그림 6. Neo4j와 SQL Graph 질의의 비교
Fig. 6. Neo4j and SQL graph query comparison

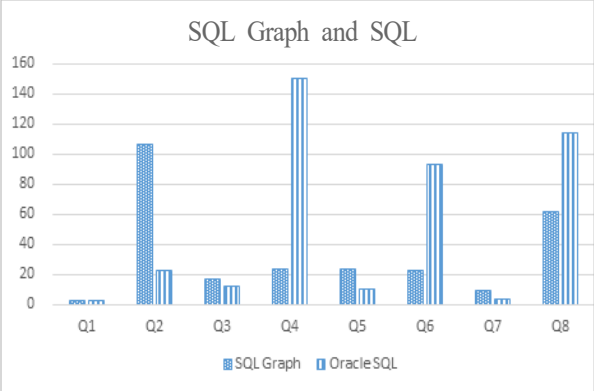

그림 7. SQL Graph와 SQL 질의의 비교
Fig. 7. SQL graph and SQL query comparison

(2) SQL Server의 SQL Graph와 오라클의 SQL 문을 비교해보면 복잡한 그래프 패턴 질의가 아닌 경우 오라클의 SQL은 SQL Server의 SQL Graph 보다 수행 시간이 작다. 오라클은 이미 알려진 성숙한 최적화 기능을 통하여 수행 시간을 단축시킨다. 그러나 그래프의 경로가 4 이상인 질의의 경우는 오라클은 Neo4j나 SQL Graph보다 수행 시간이 더 오래 걸린다(Q4, Q6, Q8). 그림 7은 SQL Graph와 오라클 SQL의 수행 시간을 각각 비교한 것이다.

전체적으로 그래프 패턴 질의에서 그래프 경로가 길어지면 MATCH 문을 사용하는 Cypher와 SQL Graph를 사용하는 질의가 일반 SQL 질의보다 성능이 뛰어나다. Cypher 질의어의 경우 탐색할 에지와 노드가 질의문 안에 바운드 되는 경우나 싸이클 질의 경우 SQL Graph 보다 더 우수한 결과를 나타내었다고 볼 수 있다.

3.3 질의 언어 비교

그래프 패턴질의를 위한 질의 언어 3 가지를 다음으로 비교할 수 있다. 질의 언어의 질적인 면은 정량화하기는 어렵지만 개발자에게 질의 언어의 선택에 도움을 줄 수 있다.

(1) 프로그래밍의 용이성 측면에서 Neo4j의 Cypher는 그래프 데이터베이스에 가장 적합하다. 그래프 데이터베이스를 사용할 수 없다면 SQL Server의 SQL Graph와 같이 그래프 질의에 SQL 언어를 사용할 수 있음은 큰 장점이다. 개발자들이 기존의 SQL 언어에 그래프 언어인 Cypher 스타일 언어를 사용함으로써 프로그래밍을 쉽게 한다. 그래프 패턴 질의를 오라클 경우와 같이 SQL 만으로 표현하면 많은 조인이 필요하다.

(2) DBMS 성숙도 및 지원 면에서는 오라클과 SQL Sever가 DBMS 기능 완성도 및 지원 면에서 매우 우수하다. 관계 데이터베이스는 다양한 응용에 구축되어 왔고 버그가 적으며 기능 지원이 많다. 그래프 데이터베이스는 아직 시장 규모가 작고 제작사인 Neo4j사나 wiki에서 기술적으로 지원되고 있다. SQL Graph는 2017년 추가된 기능으로 아직 완성도 및 지원은 낮은 편이다.

(3) 데이터 무결성 및 보안 측면에서 관계 데이터베이스는 트랜잭션의 ACID 성질을 지원하여 보안 방법 및 도구가 그래프 데이터베이스에 비하여 매우 우수하다.

데이터베이스의 그래프 질의어를 표 3에서 비교하였다.

표 3. 질의 언어 평가(◎:우수, ○:보통, △: 미흡)
Table 3. Query language comparison (◎: good, ○: moderate, △: insufficient)

	Neo4j Cypher	SQL Graph	Oracle SQL
Ease of program	◎	○	△
Speed(pattern)	○	○	○
Maturity/support	○	△	◎
Integrity/security	△	◎	◎

Ⅳ. 결론 및 향후 과제

그래프 데이터베이스는 대용량 그래프 응용에서 경로 질의, 복잡한 패턴 질의에 우수하다. 그렇지만 관계 데이터베이스 응용에서 발생하는 그래프 문제를 별도로 그래프 데이터베이스를 구축하여 처리하기 어려운 경우가 많다. 본 연구에서는 Neo4j의 Cypher, SQL 질의를 확장하여 Cypher를 결합한 SQL Server의 SQL Graph, 오라클의 SQL 언어, 3가지에 대하여 4가지 그래프 패턴 질의의 성능을 비교하였다.

그래프 패턴 질의가 간단한 경우 기존 관계 데이터베이스의 SQL 구문으로 그래프 데이터베이스에 비하여 좋은 성능을 발휘한다. 그래프 패턴 질의가 복잡한 경우 SQL Graph나 Cypher 질의가 일반 SQL 질의보다 더 우수하다. SQL Server의 SQL Graph는 Cypher보다 우수한 성능을 보였지만 질의어 안에 탐색할 노드나 에지가 바운드 되는 경우 Cypher가 SQL Graph 보다 더 우수한 성능을 보였다.

특히 SQL Graph는 관계 데이터베이스 백엔드에서 그래프 패턴 질의를 처리하는 새로운 방법으로 관계 데이터베이스의 성숙한 기능을 충분히 활용하면서 프로그래밍 용이성의 장점을 가지면서 적당한 질의 처리 속도로 문제를 해결할 수 있다. 대용량 그래프 데이터의 경우 기억장소 크기, 데이터 삽입 시간 등 고려해야할 다른 변수가 많아 더 연구가 필요하며, 또 SQL Graph와 Neo4j는 DBMS 최적화에서 아직 개선의 여지가 많다.

앞으로의 연구는 관계 데이터베이스에 구축된 응용에서 SQL Graph를 잘 활용하기 위해서는 관계 데이터베이스의 그래프 데이터 변환 방법 개발이 필요하며 본 연구에서 다루지 않은 그래프 분석질의 처리에 대한 성능도 실험을 통한 비교가 필요하다.

References

[1] Chad Vicknair, Michael Macias, Zhendong Zhao, Xiaofei Nan, Yixin Chen, and Dawn Wilkins, "A Comparison of a Graph Database and a Relational Database", ACMSE '10, Oxford, MS, USA, pp. 42, Apr. 2010.

[2] Shalini Batra and Charu Tyagi, "Comparative Analysis of Relational And Graph Databases", International Journal of Soft Computing and Engineering (IJSCE) Vol. 2, No. 2, pp. 509-512, May 2012.

[3] Subhrajyoti Bordoloi and Bichitra Kalita, "Designing Graph Database Models from Existing Relational Databases", International Journal of Computer Applications, Vol. 74, No. 1, pp. 25-31, Jul. 2013.

[4] Alexandra Martinez, Rodrigo Mora, Daniel Alvarado, Steve Quiros, and Gustavo López, "A Comparison between a Relational Database and a Graph Database in the context of a Personalized Cancer Treatment Application", Proceedings of the 10th Alberto Mendelzon International Workshop on Foundations of Data Management, Jun. 2016.

[5] Neo4j, https://neo4j.com/, [accessed: Dec. 1, 2018]

[6] Graph Database by SQL, https://gianniceresa.com/2018/03/graph-database-by-sql/, [accessed: Dec. 1, 2018]

[7] Graph processing with SQL Server and Azure SQL Database, https://docs.microsoft.com/ko-kr/sql/

relational-databases/graphs/sql-graph-overview?view=sql-server-2017, [accessed: Dec. 1, 2018]

[8] Y. Guo, Z. Pan, and J. Heflin, "An Evaluation of Knowledge Base Systems for Large OWL Datasets", in Proc. of the 3rd International Semantic Web Conference (ISWC2004), pp. 274-288, Nov. 2004.

[9] Andrey Gubichev and Manuel Then, "Graph Pattern Matching – Do We Have to Reinvent the Wheel?", GRADES'14, Snowbird, UT, USA, pp. 22-274, Jun. 2014.

[10] Jürgen Hölsch, Tobias Schmidt, and Michael Grossniklaus, "On the Performance of Analytical and Pattern Matching Graph Queries in Neo4j and a Relational Database", Workshop Proceedings of the EDBT/ICDT 2017 Joint Conference, Venice, Italy, pp. 21-24, Mar. 2017.

[11] Adam Welc, Raghavan Raman, Zhe Wu, Sungpack Hong, Hassan Chafi, and Jay Banerjee, "Graph Analysis – Do We Have to Reinvent the Wheel?" Proceedings of the First International Workshop on Graph Data Management Experience and Systems (GRADES 2013), New York, NY, USA, Jun. 23, 2013.

[12] Michael Grossniklaus, Stefania Leone, and Tilmann Zäschke, "Towards a Benchmark for Graph Data Management and Processing", University of Konstanz Department of Computer and Information Science Technical Report KN-2013-DBIS-01, Jan. 2013.

[13] Yaowen Chen, "Comparison of Graph Databases and Relational Databases When Handling Large-Scale Social Data", Master Thesis, Department of Computer Science, University of Saskatchewan, Sep. 2016.

[14] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin, "An Evaluation of Knowledge Base Systems for Large OWL Datasets", ISWC 2004, pp. 274-288, Nov. 2004.

[15] Roman Kontchakov, Martín Rezk, Mariano Rodríguez-Muro, Guohui Xiao, and Michael Zakharyaschev, "Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime", ISWC 2014, pp. 552-567, Oct. 2014.

저자소개

박 우 창 (Uchang Park)

1982년 2월 : 서울대학교 계산통계학과(이학사)
1985년 2월 : 서울대학교 계산통계학과(이학석사)
1993년 2월 : 서울대학교 계산통계학과(이학박사)
1988년 9월 ~ 현재 : 덕성여자대학교 컴퓨터공학과 교수
관심분야 : 데이터베이스 질의어, 데이터마이닝