软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(3):781−804 [doi: 10.13328/j.cnki.jos.006181]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

KGDB:统一模型和语言的知识图谱数据库管理系统*

刘宝珠[1,2], 王 鑫[1,2], 柳鹏凯[1,2], 李思卓[1,2], 张小旺[1,2], 杨雅君[1,2]

[1](天津大学 智能与计算学部,天津 300354)

[2](天津市认知计算与应用重点实验室,天津 300354)

通讯作者: 王鑫, E-mail: wangx@tju.edu.cn

摘 要: 知识图谱是人工智能的重要基石,其目前主要有 RDF 图和属性图两种数据模型,在这两种数据模型之上有数种查询语言.RDF 图上的查询语言为 SPARQL,属性图上的查询语言主要为 Cypher.10 年来,各个社区开发了分别针对 RDF 图和属性图的不同数据管理方法,不统一的数据模型和查询语言限制了知识图谱的更广泛应用.KGDB(knowledge graph database)是统一模型和语言的知识图谱数据库管理系统:(1) 以关系模型为基础,提出了统一的存储方案,支持 RDF 图和属性图的高效存储,满足知识图谱数据存储和查询负载的需求;(2) 使用基于特征集的聚类方法解决无类型实体的存储问题;(3) 实现了 SPARQL 和 Cypher 两种不同知识图谱查询语言的互操作性,使其能够操作同一个知识图谱.在真实数据集与合成数据集上进行的大量实验表明:KGDB 与已有的知识图谱数据库管理系统相比,不仅能够提供更加高效的存储管理,而且具有更高的查询效率.KGDB 平均比 gStore 和 Neo4j 节省了 30%的存储空间,基本图模式查询上的实验表明:在真实数据集上的查询速度普遍高于 gStore 和 Neo4j,最快可提高 2 个数量级.

关键词: 知识图谱;SPARQL;Cypher;RDF 图;属性图

中图法分类号: TP311

中文引用格式: 刘宝珠,王鑫,柳鹏凯,李思卓,张小旺,杨雅君.KGDB:统一模型和语言的知识图谱数据库管理系统.软件学报, 2021,32(3):781−804. http://www.jos.org.cn/1000-9825/6181.htm

英文引用格式: Liu BZ, Wang X, Liu PK, Li SZ, Zhang XW, Yang YJ. KGDB: Knowledge graph database system with unified model and query language. Ruan Jian Xue Bao/Journal of Software, 2021,32(3):781−804 (in Chinese). http://www.jos.org.cn/1000-9825/6181.htm

KGDB: Knowledge Graph Database System with Unified Model and Query Language

LIU Bao-Zhu[1,2], WANG Xin[1,2], LIU Peng-Kai[1,2], LI Si-Zhuo[1,2], ZHANG Xiao-Wang[1,2], YANG Ya-Jun[1,2]

[1](College of Intelligence and Computing, Tianjin University, Tianjin 300350, China)

[2](Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin 300350, China)

Abstract: Knowledge graph is an important cornerstone of artificial intelligence, which currently has two main data models: RDF graph and property graph. There are several query languages on these two data models. The query language on RDF graph is SPARQL, and the query language on property graph is mainly Cypher. Over the last decade, various communities have developed different data management methods for RDF graphs and property graphs. Inconsistent data models and query languages hinder the wider application of knowledge graphs. KGDB is a knowledge graph database system with unified data model and query language. (1) Based on the relational model, a unified storage scheme is proposed, which supports the efficient storage of RDF graphs and property graphs, and meets the requirement of

* 基金项目: 国家重点研发计划(2019YFE0198600); 国家自然科学基金(61972275); CCF-华为数据库创新研究计划(CCF-Huawei DBIR2019004B)

Foundation item: National Key Research and Development Program (2019YFE0198600); National Natural Science Foundation of China (61972275); CCF-Huawei Database Innovation Research Plan (CCF-Huawei DBIR2019004B)

本文由“支撑人工智能的数据管理与分析技术”专刊特约编辑陈雷教授、王宏志教授、童咏昕教授、高宏教授推荐.

收稿时间: 2020-07-20; 修改时间: 2020-09-03; 采用时间: 2020-11-06; jos 在线出版时间: 2021-01-21

knowledge graph data storage and query load. (2) Using the clustering method based on characteristic sets, KGDB can handle the issue of untyped triple storage. (3) It realizes the interoperability of SPARQL and Cypher, which are two different knowledge graph query languages, and enables them to operate on the same knowledge graph. The extensive experiments on real-world datasets and synthetic datasets are carried out. The experimental results show that, compared with the existing knowledge graph database management systems, KGDB can not only provide more efficient storage management, but also has higher query efficiency. KGDB saves 30% of the storage space on average compared with gStore and Neo4j. The experimental results on basic graph pattern matching query show that, for the real-world dataset, the query efficiency of KGDB is generally higher than that of gStore and Neo4j, and can be improved by at most two orders of magnitude.

Key words: knowledge graph; SPARQL; Cypher; RDF graph; property graph

知识图谱是人工智能的重要基石,是新一代人工智能由感知走向认知的重要基础设施[1].RDF 图和属性图是知识图谱的两个主流数据模型.一方面,资源描述框架(resource description framework,简称 RDF)成为国际万维网联盟(W3C)表示知识图谱的推荐标准,被以 gStore[2]为代表的三元组数据库广泛采用;另一方面,属性图被以 Neo4j[3],Dgraph[4]和 HugeGraph[5]等为代表的图数据库广泛采用为底层数据模型.

关系数据库数十年来的发展,证实了统一的数据模型和查询语言是数据管理技术发展的关键.目前,知识图谱数据库管理的问题是数据模型、存储方案和查询语言不统一.为此,我们研发了统一模型和语言的知识图谱数据库管理系统——KGDB(knowledge graph database)

KGDB 使用统一的存储方案,可以支持存储 RDF 图和属性图两种不同的数据模型;根据实体的类型进行分块存储,同时运用基于特征集的聚类方法处理无类型实体,使之可以归入某一语义相近的类型;分别提供 RDF 图和属性图上的查询接口,可以使用 SPARQL 和 Cypher 查询语言对同一知识图谱进行操作,即允许两种查询语言的互操作.KGDB 遵循“统一存储”-“兼容语法”-“统一语义”的技术路线:在底层存储,使用相同的存储方案处理不同的知识图谱数据模型;在查询表达上,兼容两种面向不同知识图谱数据模型的语法完全不同的查询语言;而在查询处理上,将两种语法不同的查询语言对齐到统一的语义,进而使用相同的查询处理方法.

KGDB 的总体架构如图 1 所示,采用自底向上的系统构建方法.

(1) 在用户输入层,用户可输入 RDF 图数据或者属性图数据;

(2) 在系统处理阶段,分为两部分:①经过存储关系转化,依据存储方案将数据转化为以类型聚类的关系表,将原始的知识图谱数据进行基于关系的存储;②查询处理部分,可以允许用户使用两种查询语言对同一知识图谱进行操作;

(3) 在用户界面层,用户可以查看规范格式的图模式查询的结果.

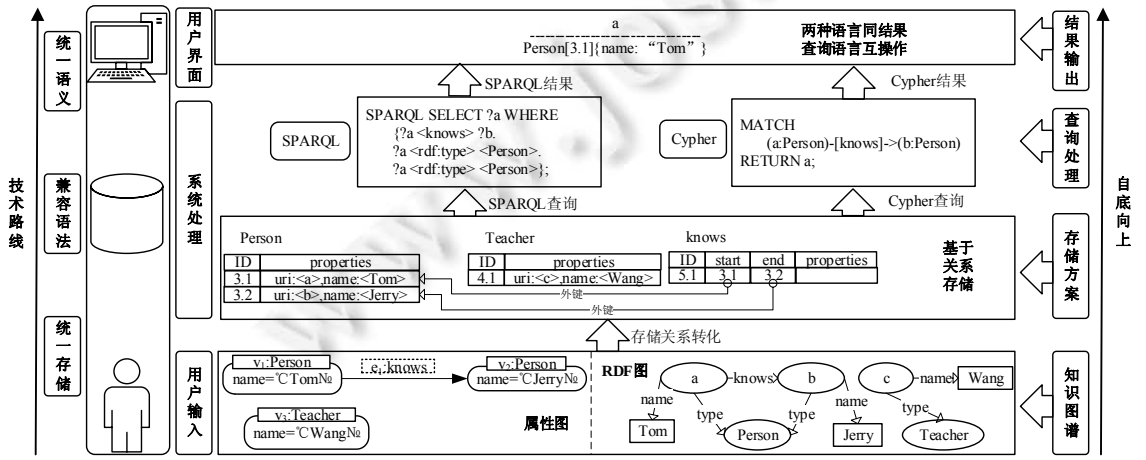

Fig.1 Architecture of KGDB

图 1 KGDB 总体架构图

现有的知识图谱数据库管理系统因其面向的数据模型的不同,提出了不同的存储方法.对于RDF图的存储,可以分为三大类:(1)直接利用RDF三元组的特性进行存储,例如三元组表(triple table)、水平表(horizontal table)等;(2)根据RDF图数据展现的数据类型特征进行存储,例如属性表(property table)、垂直划分(vertical partitioning)[6]、六重索引(sextuple indexing)和DB2RDF[7]等;(3)依据RDF图数据的语义信息进行归类存储,例如特征集方法(characteristic sets)[8]、扩展的特征集方法(extended characteristic sets)[9]和R-Type[10]等.对于属性图的存储,一般采用生存储方案,例如Neo4j,JanusGraph[11],TigerGraph[12]等.

知识图谱三元组数据与关系数据的最大不同是其模式的灵活性,这给传统的存储和查询处理提出了新问题.为此,提出了一种基于特征集聚类的处理方法,根据实体的谓语组进行类型聚类,并将无类型实体数据依据其关系特征,归入某类型中,从而解决知识图谱的统一存储问题.

在查询语言方面,现有的知识图谱数据库管理系统因面向的数据模型不同,使用不同的查询语言进行查询处理:SPARQL是RDF图上的查询语言,Cypher是属性图上的主要查询语言.两种查询语言具有不同的语法,阻碍了在统一存储方案上的查询互操作.为此,进行SPARQL和Cypher语言的语义对齐,使之可以操作同一个知识图谱,而无需区别知识图谱的数据模型.

真实数据集和合成数据集上的大量实验表明:KGDB能够高效地进行知识图谱数据的存储管理和查询处理,优于现有RDF数据库gStore和属性图数据库Neo4j.

本文的主要贡献如下:

(1) 以关系模型为基础,提出统一的知识图谱存储方案,根据数据的类型进行聚类,支持两种知识图谱主流数据模型(RDF图和属性图)的高效存储;使用字典编码方法减少数据的存储空间,满足知识图谱数据的存储和查询负载需求;

(2) 使用基于特征集的聚类方法,将无类型实体归入谓语组相似的数据类型中,解决无类型实体数据的存储问题,使统一存储模型能够应对灵活多变的半结构化数据;

(3) 兼容RDF图数据模型的查询语言SPARQL和属性图数据模型的主流查询语言Cypher的查询语法,进行两种查询语言的语义对齐,实现两种查询语言的互操作,可使用两种语言操作同一个知识图谱;

(4) 在合成数据集和真实数据集上进行大量实验,分别与现有的RDF图数据管理系统和属性图数据管理系统进行对比实验,实验结果验证了KGDB统一的存储方案和统一语义的查询处理的有效及高效性.

本文第1节介绍相关工作.第2节介绍预备知识.第3节描述KGDB使用的统一RDF图和属性图的存储模型.第4节阐述查询语言互操作的方法.第5节在真实数据集和合成数据集上进行实验.第6节对全文总结.

1 相关工作

随着知识图谱的发展,各种知识图谱存储方案和查询处理方法不断涌现,本节将分别介绍两种知识图谱数据模型的现有存储方案及查询方法.知识图谱数据规模的不断增长,对存储和查询提出更高要求,分布式知识图谱管理系统成为研究热点之一[13,14].

1.1 知识图谱存储方案

1.1.1 RDF图数据存储

(1) 直接利用RDF三元组特性

三元组表(triple table)将数据对应存储到一个3列结构的表中,采用三元组表存储方案的代表是3store[15].水平表(horizontal table)在每一行存储一个主语的所有谓语及对应的宾语,DLDB[16]系统采用了水平表存储方案.属性表(property table)将同类的主语在一个表中存储,Jena[17]采用了属性表存储方案.垂直划分(vertical partitioning)[6]对每一个谓语建立一个两列的表,存储该谓语连接的主语和相应的宾语.采用垂直划分存储方案的系统有SW-Store [18],TripleBit[19].六重索引(sextuple indexing)应用"空间换时间"策略,存储三元组的全部6种排列,并在首列上建立索引.使用六重索引的系统有RDF-3X[20]和Hexastore [21].DB2RDF[7]存储方案不将表的列与某一谓语绑定,而是通过哈希的方式进行动态映射,并确保将相同谓语映射到同一列上,并通过额外的表处理

多值谓语的问题.IBM DB2 数据库系统采用了 DB2RDF 存储方案.

直接利用 RDF 三元组特性的存储方案最直观简单,但是存在着诸如稀疏性、空值等空间利用的问题.同一主语对应的谓语可能会有很多,不同主语之间关联的谓语差异远高于预期,即使同一类型的主语其谓语差异也不容忽视,这类存储方案建立的关系表会有很多位置存为空值,稀疏的关系表严重降低存储性能.

(2) 依据 RDF 数据语义信息

特征集(characteristic sets,简称 CS)[8]存储方法将 RDF 数据按照星型结构进行划分,具有相同谓语组的实体将会作为同一类型对待,大大减少了需要建立的表的数量.然而,特征集方法等同地对待所有谓语,很有可能导致大多数实体落入同一集合,不能很好地达到划分目的.RDF-3X 系统实现了特征集方法[20].扩展的特征集(extended characteristic sets,简称 ECS)[9]方法实施多层次的星型结构划分,形成二级索引,加速查询.但是,扩展的特征集方法也不能避免大多数实体落入同一集合的问题.R-Type[10]方法引入了本体规则,将谓语分为领域可推定的和非领域可推定的,并将包含领域可推定谓语的星型团结构中指定类型的三元组省略,不进行物理化存储,节省存储空间.在查询过程中,包含领域可推定的星型结构可以直接映射到正确的团结构上,从而加速查询.但 R-Type 没有对无类型实体提出划分方法.SemStorm[22]系统采用了 R-Type 方法.

依据 RDF 数据语义信息的存储方案虽然更加精确,能够利用语义信息优化存储,但是目前没有见到基于关系的存储方案,且大多数方案仅有原型系统,成熟度不足.关系数据库发展至今,在事务管理、可扩展性等方面的研究基础相对稳固,基于关系的存储方案可以获得更多的支持.

1.1.2 属性图数据存储

(1) 基于关系的存储方案

SQLGraph[23]使用一种在关系数据库中利用关系和 JSON 键值存储属性图的方案,将每个边标签散列到两列,将边的邻接列表存储在关系表中,其中一列存储边标签,而另一列存储值.AgensGraph[24]是一种底层基于关系模型的多模型图数据库,将属性图中的点和边根据标签分开存储到关系表中,并将点、边的属性值信息以 JSON 格式进行记录.

因属性图的半结构化数据特征,上述基于关系的属性图存储方案的灵活度不能完全满足属性图的存储要求,关系表一旦建立,修改其结构的代价巨大.而对于如今知识图谱高达数十亿点边结构的数据规模,需要更加灵活的方案来提高其存储效率.

(2) 基于文档的存储方案

MongoDB[25]是一种基于文档存储的数据库系统,旨在为 Web 应用提供可扩展的高性能数据存储解决方案,将数据存储为一个文档.数据结构由键值对组成,其文档类似于 JSON 对象,字段值可以进行嵌套.Neo4j[3]将所有数据存储在节点和关系中,不需要任何额外的关系数据库或 NoSQL 数据库来存储数据,以图的原生形式存储属性图数据并应对各种查询需求.

基于文档的属性图存储方案常被应用在分布式环境下,而相对于单机环境,分布式对数据的存储提出了更高的要求,而使用文件进行大规模数据的存储效率不足以满足数据规模日益增长的属性图的存储和查询要求.目前,知识图谱的存储方案基本都面向某一种类型的知识图谱并且成熟度不足.为此,有必要面向知识图谱的两种主流数据模型开发统一的、基于关系的高效存储方案.

1.2 知识图谱查询处理

1.2.1 RDF 图数据查询

Blazegraph[26]是一个基于 RDF 三元组库的图数据库管理系统,其内部实现技术是面向 RDF 三元组和 SPARQL 查询语言的.Jena[17]是语义 Web 领域主要的开源框架和 RDF 三元组库,较好地遵循了 W3C 标准,支持 SPARQL 查询处理,具有一套基于规则的推理引擎,用以执行 RDFS 和 OWL 本体推理任务.gStore[2]使用 RDF 图对应的签章图并建立 VS 树索引,支持 SPARQL 查询处理.Virtuoso[27]是支持多种数据模型的混合数据库管理系统,可以较为完善地支持 W3C 的 Linked Data 系列协议.RDF4J[28]前身是 Aduna 公司开发的 Sesame 框架,支持 SPARQL 1.1 的查询和更新语言,能够进行 RDF 数据的解析、存储、推理和查询.RDF-3X[29]为 RDF 数据专门设

计了压缩物理存储方案、查询处理和查询优化技术.AllegroGraph[30]系统对语义推理功能具有较为完善的支持.GraphDB[31]实现了 RDF4J 框架的 SAIL 层,使用内置的基于规则的“前向链(forward-chaining)”推理机,对 RDF 推理功能拥有良好的支持.

1.2.2 属性图数据查询

Neo4j[3]是目前流行程度最高的图数据库产品,基于属性图模型,支持 Cypher 查询语言.AgensGraph[24]基于关系模型存储属性图,在 PostgreSQL 内核之上构建 Cypher 语言的处理层.JanusGraph[11]是开源分布式图数据库,存储后端与查询引擎分离,具备基于 MapReduce 的图分析引擎,可将 Gremlin[32]导航查询转化为 MapReduce 任务.OrientDB[33]主要面向图和文档数据存储管理的需求设计,支持扩展的 SQL 和 Gremlin 用于图上的导航式查询,支持类似 Cypher 语言查询模式的 MATCH 语句.Cypher for Apache Spark[34]提供基于 Spark 框架的 Cypher 引擎.

目前,两种知识图谱数据模型拥有各自的查询语言、语法和语义,虽然在具体的系统上分别进行了优化设计,但不利于知识图谱查询的多样性,故亟需研发一种既支持 SPARQL 又支持 Cypher、具有语义互操作能力的系统.

2 预备知识

本节将详细介绍相关背景知识,包括 RDF 图和属性图的定义.表 1 给出了本文使用的主要符号及其含义.

Table 1 List of notations

表 1 符号列表

符号	含义	符号	含义
$G=(V,E,\Sigma)$	RDF 图 G	$\varphi(t)$	三元组分类
$G=(V,E,\eta,src,tgt,\lambda,\gamma)$	属性图 G	$I_C(s)$	实体特征集
$t=(s,p,o)$	三元组	$hist(C)$	实体簇统计信息
$\alpha=(a,Lab,Map)$	属性图点模式	$DCS_{cluster}(C_j,C_k)$	两个实体簇 C_j 和 C_k 的距离
$\beta=(d,Lab,a,Map)$	属性图边模式	$\lambda:(V\cup E)\to\mathcal{L}$	属性图顶点、边到标签映射
T	三元组有限集合	$\gamma:(V\cup E)\times\mathcal{K}\to Val$	属性到属性值的映射
μ	匹配	$lab:E\to\Sigma$	获取 RDF 图中边的标签

定义 1(RDF 图).设 U 是统一资源标识符的有限集合,L 是字面量的有限集合,B 是空结点的有限集合.元组 $t=(s,p,o)\in U\times U\times(U\cup L\cup B)$称为 RDF 三元组(在本文中不考虑实体为空结点的情况),三元组 $t=(s,p,o)$表示资源 s 和资源 o 有关系 p,或者资源 s 的属性 p 的值为 o.其中,s,p 和 o 分别表示主语、谓语和宾语.RDF 图 G 是三元组 t 的有限集合.用 V,E,Σ分别表示 RDF 图 G 的顶点、边和标签的集合.正式定义为:$V=\{s|(s,p,o)\in G\}\cup\{o|(s,p,o)\in G\}$,$E\subseteq V\times V$ 且 $\Sigma=\{p|(s,p,o)\in G\}$.函数 $lab:E\to\Sigma$返回图 G 中边的标签.

例 1:图 2 所示的 RDF 图示例描述了一个音乐知识图谱,包括作曲家(Composer)Beethoven、钢琴演奏家(Pianist)Lang Lang 和音乐(Music)Fate Symphony 等资源,这些资源上的若干属性以及资源之间的作曲(composes)和演奏(plays)联系.在 RDF 图示例中.使用椭圆表示资源,矩形表示字面量,连接顶点之间的有向线段表示顶点之间的关系,起点是主语,边标签是谓语,终点是宾语.RDF 内置谓语 rdf:type 指定资源所属类型,如三元组(Beethoven,rdf:type,Composer)表示 Beethoven 的类型是作曲家.

定义 2(属性图). 属性图 $G=(V,E,\eta,src,tgt,\lambda,\gamma)$,其中:$V$ 表示顶点有限集合;E 表示边的有限集合且 $V\cap E=\varnothing$;函数 $\eta:E\to(V\times V)$表示边到顶点对的映射,如 $\eta(e)=(v_1,v_2)$表示顶点 v_1 与顶点 v_2 之间存在一条有向边 e;函数 $src:E\to V$ 表示边到起始顶点的映射,例如 $src(e)=v$ 表示边 e 的起始顶点为 v;函数 $tgt:E\to V$ 表示边到终结顶点的映射,例如 $tgt(e)=v$ 表示边 e 的终结顶点为 v;函数$\lambda:(V\cup E)\to\mathcal{L}$为顶点或边与标签的映射(其中,$\mathcal{L}$表示标签集合),如 $v\in V$(或 $e\in E$)且$\lambda(v)=l$(或$\lambda(e)=l$),则 l 为顶点 v(或边 e)的标签;函数$\gamma:(V\cup E)\times\mathcal{K}\to Val$(其中,$\mathcal{K}$为属性集合,$Val$ 是值的集合)表示顶点或边的关联属性,如 $v\in V$(或 $e\in E$),$property\in\mathcal{K}$且$\gamma(v,property)=val$(或$\gamma(e,property)=val$)表示顶点 v(或边 e)上属性 $property$ 的值为 val.

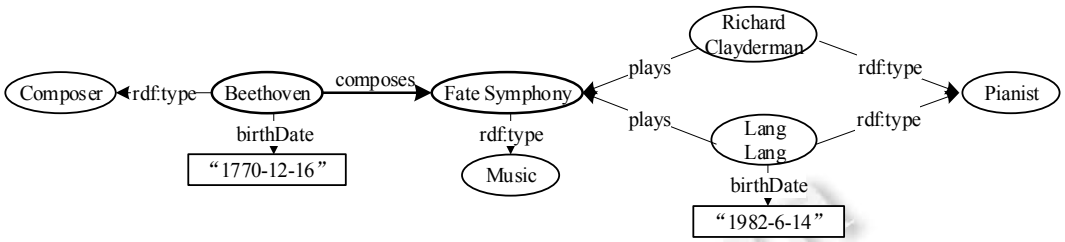

Fig.2 An RDF graph example

图 2 RDF 图示例

例 2:图 3 给出了图 2 中音乐知识图谱的属性图示例.属性图中每个顶点和边都具有唯一 id(如顶点 v_1、边 e_1),顶点和边均可具有标签(如顶点 v_1 上的标签 Composer),顶点和边上均可具有属性,每个属性由属性名和属性值的键值对组成(如顶点 v_1 上的属性 name="Beethoven").

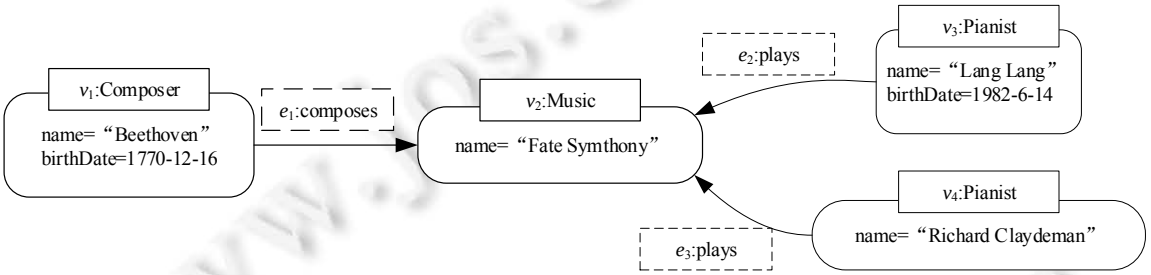

Fig.3 A property graph example

图 3 属性图示例

3 知识图谱存储方案

本节主要介绍统一的知识图谱存储方案,此方案基于关系模型实现对 RDF 图和属性图的兼容表示.之后,根据所提出的存储方案,采用基于特征集的聚类算法处理无类型数据,以更好地支持知识图谱数据存储..

3.1 知识图谱存储模型

统一存储方案按照实体的类型将其存储到对应顶点类型表 v_n 中($n\in[1,i]$),按照关系的类型将其存储到对应边类型表 e_m($m\in[1,j]$)中,其中,i,j 分别表示顶点和边类型的数量.不同的关系表以实体或关系的类型来命名.用于存储实体的关系表包含 2 列,分别存储实体的唯一标识 id(主键)和实体所拥有的属性 property(由属性名和属性值的键值对所构成).用于存储实体间关系的关系表包含 4 列,分别存储边的唯一标识 id(主键)、边的起始顶点标识 start、终结顶点标识 end 以及边所拥有的属性 property(由属性名和属性值的键值对所构成).顶点类型表和边类型表根据知识图谱数据中实体和关系的类型进行进一步地划分,同类型的顶点或者边存储在同一个关系表中,这样避免了因单个关系表数据量过大而导致的数据访问性能较低的问题.

3.1.1 RDF 图到统一存储模型的映射

RDF 图数据中有 3 种不同类型的三元组,下面给出三元组分类的定义.

定义 3(三元组分类).令 $C=\{mem,prop,edge\}$表示三元组的类别,分别指类型成员三元组、属性描述三元组和动作三元组.函数φ:$T\to C$ 是三元组到三元组类别的映射:

$$\varphi(t)=\begin{cases} mem, & 当t\in\{t=(s,p,o)\mid(s,p,o)\in T\wedge p=\text{rdf:type}\} \\ prop, & 当t\in\{t=(s,p,o)\mid(s,p,o)\in T\wedge o\in L\} \\ edge, & 当t\in\{t=(s,p,o)\mid(s,p,o)\in T\wedge p\neq\text{rdf:type}\} \end{cases} \tag{1}$$

根据定义 3 以及例 1 和例 2,将图 2 和图 3 中的事实以三元组的形式表示，并应用三元组分类方法,可知

φ((Beethoven,rdf:type,Composer))=*mem*,φ((Beethoven,birthDate,1770-12-16))=*prop* 且 φ((Lang Lang,plays,Fate Symphony))=*edge*.

对于 RDF 三元组 $t=(s,p,o)$,需要根据 RDF 图 G 中顶点标签和边标签创建相应的顶点类型表和边类型表.根据三元组的不同形式,使用下面的转换规则将其分别进行映射,将不同的实体和关系存储到相应的顶点类型表和边类型表中.

规则 1. 对于任意三元组 $t=(s,p,o)$,若$\varphi(t)=mem$,则将 id 为 s 的元组插入到表名为 o 的顶点类型表中.

规则 2. 对于任意三元组 $t=(s,p,o)$,若$\varphi(t)=prop$,则将 p,o 以键值对的形式插入到顶点类型表中实体 s 对应的 property 列中.

规则 3. 对于任意三元组 $t=(s,p,o)$,若$\varphi(t)=edge$,则在表名为 p 的边类型表中插入一条 start 为 s、end 为 o 的记录.

3.1.2 属性图到统一存储模型的映射

属性图因其对顶点和边上的属性提供内置的支持,在将其映射到统一存储模型时相对容易,应用下列规则将属性图数据映射到统一存储模型上.

规则 4. 对于属性图数据中的实体,根据其顶点标签$\lambda(v)$,为实体 v 赋唯一的数值 id 并插入到顶点类型表$\lambda(v)$的 id 列,同时将其属性 *property* 和属性值$\gamma(v,property)$的键值对插入到 property 列中.

规则 5. 对于属性图数据中的关系,根据其边标签$\lambda(e)$,为关系 e 赋唯一的数值 id 并插入到边类型表$\lambda(e)$的 id 列中,同时将其属性 *property* 和属性值$\gamma(e,property)$的键值对插入到 property 列中,将顶点 v_1 的 id 插入到 start 列中,将顶点 v_2 的 id 插入到 end 列中(其中,$\eta(e)=(v_1,v_2)\wedge src(e)=v_1\wedge tgt(e)=v_2$).

在属性图中,顶点关系表中的 id 值仅起到标识作用,而不具有实际意义;而 RDF 图中 id 表示对应实体的 URI,具有实际意义.为了进行统一的表示,将实体 v 的 URI 值 v_{uri} 作为一个新的属性,添加到结点关系表中的 property 列中,即添加$\gamma(v,uri)=v_{uri}$.

例 3:根据上述的存储方案,将图 2、图 3 介绍的音乐知识图谱示例进行相应转换,得到图 4 基于关系的统一知识图谱存储方案.不同的实体按照其类型(作曲家、钢琴家、音乐)存储到顶点类型表中,不同的关系按照类型(作曲、演奏)存储到关系类型表中.图中箭头表示外键关系.

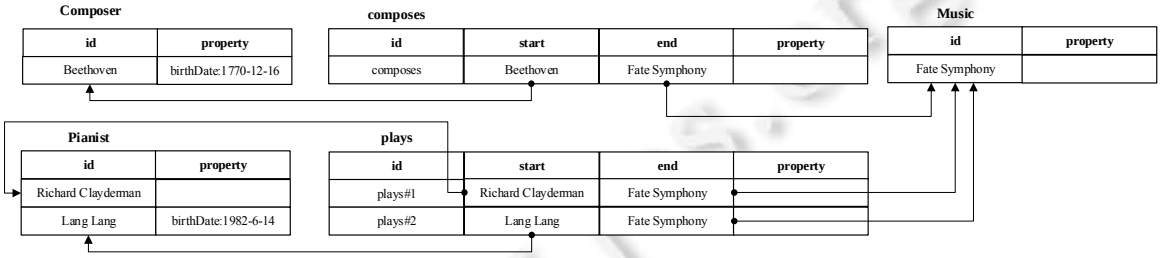

Fig.4 Unified storage scheme for knowledge graph

图 4 知识图谱统一存储方案

知识图谱数据根据所提出的统一存储方案,依据实体和关系的类型进行分别存储.在示例的知识图谱中未给出边属性,因此边表中的 property 列为空值.属性图和 RDF 图对实体或者关系的类型信息均提供了内置的支持,属性图由标签指定,RDF 图由内置的 rdf:type 指定.这种根据类型对顶点和边进行划分并进行分别存储管理的方式是相对合理的,能够在一定程度上解决现有知识图谱存储方案中存在的数据冗余与数据稀疏问题.

在建立关系表之后,各种操作都将以关系表名为基本对象,给出操作关系表的函数:存储模型建立的关系表集合是 $R=\{r_1,r_2,\ldots,r_n\}$,对应的关系表的表名集合是 $X=\{x_1,x_2,\ldots,x_m\}$;函数 $name:R\to X$ 返回某个关系表的表名;函数 $rel:T\to R$ 返回某个实体 t 所在的关系表,其中,T 为实体集合且 $t\in T$.

在知识图谱中,两节点之间有可能存在多种谓语关系,即多重属性.对于多重属性的存储问题,KGDB 使用第

3.1 节中介绍的方法,针对单个主语实体,存储多个属性键值对,其中,属性键对应不同的谓语关系,属性值对应同一个实体.大多数现有的知识图谱数据存储方案使用字典编码的方法,将 URI 或者字面量映射到整数标识符,即 id.映射表有效地实现了字符串到 id 之间的转换,并将数据库的空间开销降至最低.KGDB 采用了与大多数现有知识图谱存储方案类似的字典编码方法,压缩存储方案所需的空间资源.

3.2 无类型实体存储优化方案

上一节介绍的统一存储模型中,利用顶点和边的类型对知识图谱数据进行划分,并存储到相应的顶点表和边表中.但此方案并没有考虑无类型的实体的存储.对于无类型的实体,基本的存储方案将所有无类型的实体等同对待,所有无类型的实体存储在一个关系表中.这种统一存储的方法,一方面在处理无类型实体数量较多的数据集时,将导致单个关系表过大,降低查询的效率;另一方面,无类型的实体之间并没有关联,无语义信息,这与将相同或靠近语义的实体存储在相近空间的初衷相悖.

对于未指定类型的实体,基于特征集的聚类算法,将其划分到一个最相近的给定类别中.层次聚类算法可以根据某种距离函数给出结点之间的相似性,并按照相似度逐步将结点进行合并,当达到某一条件时,合并终止.在本节中,将给出实体的特征集、特征集的距离等定义,用于测定实体之间的相似性,从而解决无法对无类型实体进行存储的问题.

3.2.1 实体特征集

RDF 图中使用多个三元组来描述一个实体所拥有的特征.实体的特征集[8]可定义为 RDF 图中表示该实体的顶点所发出的边的名称的集合.下面给出实体特征集的形式化定义.

定义 4(实体特征集). 对于每个出现在知识图谱数据集 D 中的实体 s,定义其特征集 I 如下:

$$I_C(s)=\{p|\exists o:(s,p,o)\in D\} \tag{2}$$

例 4:在某个 RDF 数据集中,描述《老人与海》这本书的实体 s_1 共使用了 3 个三元组,分别为(s_1,title,“The Old Man and the Sea”),(s_1,author,Hemingway)以及(s_1,year,“1951”),则 s_1 的特征集是 $I_C(s_1)$={title,author,year}.

3.2.2 实体簇及其距离定义

实体簇 C 中包含多个实体,为了更好地表示一个簇中所包含实体的属性的特征,对任意一个包含若干个实体的簇 C,定义 $hist(C)$来表示簇中实体的特征集的统计信息,记录簇中包含的全部实体所拥有的属性的个数为 m,则 $hist(C)$可以定义为每个属性与它出现次数的键值对的集合:

$$hist(C)=\bigcup_{i=1}^{m}(property_i,count_i) \tag{3}$$

其中,$property_i$ 表示第 i 个属性,$count_i$ 表示第 i 个属性在簇 C 中出现的次数.进而可以通过这一统计信息定义两个包含若干实体的簇的距离:对于两个簇 C_j 和 C_k,它们之间的距离可以表示为其实体统计信息之间的距离,即:

$$DCS_{cluster}(C_j,C_k)=DCS_{hist}(hist(C_j),hist(C_k))=\sum_{p_i\in\{p_1,\ldots,p_n\}}(count(C_j,p_i)+count(C_k,p_i))b_i \tag{4}$$

其中,

- n 是两个簇中所含不同属性的总个数;
- b_i 表示第 i 个属性是否同时出现在两个簇中:若是,则为 0;否则为 1;
- $count(C_j,p_i)$和 $count(C_k,p_i)$分别表示在簇 C_j 和 C_k 中,第 i 个属性 p_i 对应的个数.

根据公式(4),两个簇之间的距离等于不同时出现在两个簇中的属性所对应的出现次数之和.属性的出现次数可以说明该属性对簇的重要程度,如对于书籍类型的实体来说,其所在的簇中作者和标题属性所对应的数值较高.以属性次数的加和作为簇间距离,可以衡量两个簇之间的相似程度.

3.2.3 基于特征集的实体聚类算法

基于实体特征集、包含多个实体的簇的特征集统计信息以及簇间距离的定义,可以对基本的统一存储方案进行优化.基于层次聚类算法,提出一种基于实体类型的实体聚类算法,将未给定类型的实体通过聚类的方法划分到一个已知的类型中.

对于实体 $s\in S$,其中,S 为实体集合,函数 $haveType:S\rightarrow${TRUE,FALSE},返回实体的有无类型情况:若 s 为有类

型实体,则返回真;否则返回假.函数 *getType*:$S\rightarrow TYPE$ 返回某个实体的类型信息,其中,*TYPE* 为实体类型集合.

为了将两个实体簇进行合并,需要计算两个实体簇之间的距离,并找到距离最相近的两个实体簇进行合并.下面给出寻找距离最近实体簇下标的函数:对于实体簇集合 $C=\{C_1,C_2,\ldots,C_n\}$,函数 *findMin*(C)两两计算实体簇间距离,并给出距离最近的两个不同实体簇的下标.

在聚类的过程中,将每个实体作为一个单独的簇,并根据实体的特征集的相似程度自底向上进行簇的合并操作.合并过程中,需要保证不对两个包含不同类型实体的簇进行合并.

算法 1 给出了基于特征集的实体聚类算法,根据实体的特征集进行层次聚类,可以将实体按照所拥有的特征划分到不同的簇中,每个簇内的实体属性相似,即每个簇内的实体趋近于拥有相同的类型.算法首先将实体类型相同的实体合并到一个簇中,将每个未指定类型的实体各自作为一个单独的簇(第 2 行~第 8 行),根据簇间的距离 $DCS_{cluster}$ 进行自底向上地合并,在已知的实体簇集合 C 中找到簇间距离最小的两个簇 C_i 和 C_j,即令 $DCS_{cluster}(C_i,C_j)$的值最小,且要求这两个簇不都为已经给定类型的实体簇(第 10 行~第 12 行),合并两个簇,并将合并后的簇的类型指定为其中已知类型的簇的类型 C_i,同时更新合并后的簇的统计信息 *hist*(C_i)(第 14 行、第 15 行).重复合并簇的操作,直到没有符合条件的两个簇为止.经过实体聚类,包含未指定类型实体的簇将被合并到包含指定类型的实体的簇中,且每个簇中的实体的类型相同.

算法 1. 三元组无类型实体聚类.

输入:实体集合 S;

输出:实体簇集合 C.

```
1.  for each s∈S do
2.    if haveType(s) then
3.      τ←getType(s);          //向类型为τ的实体所在簇中添加实体 s
4.      C_τ←C_τ∪{s};
5.      C←C∪{C_τ};
6.    else
7.      C_0←C_0∪{s};           //将无类型实体作为单独簇添加到簇集合 C 中
8.      C←C∪{C_0};
9.  end
10. while |C|>1 do
11.   i,j←findMin(C);          //获取簇距离最小且不同的两个已知类型簇的下标
12.   if i=0∧j=0 then          //找不到满足条件的两个簇
13.     break;
14.   C_i←C_i∪C_j;             //将簇 C_i 和 C_j 合并
15.   C←C\C_j;
16. end
```

例 5:下面通过一个例子对聚类的过程进行说明.

在聚类开始时,rdf:type 为书籍的实体有 s_1 和 s_2,合并到簇 C_1 中,其中,

- $I_C(s_1)$={title,author,year};
- $I_C(s_2)$={author,year};
- *hist*(C_1)={(title,1),(author,2),(year,2)}.

已知 rdf:type 为电影的实体有 s_3 和 s_4,合并到簇 C_2 中,其中,

- $I_C(s_3)$={title,director,year};
- $I_C(s_4)$={director,year};
- *hist*(C_2)={(title,1),(director,2),(year,2)}.

最后,没有 rdf:type 的实体 s_5 作为一个簇 C_3:

- $I_C(s_5)$={title,director};
- $hist(C_3)$={(director,1)}.

故簇 C_1 和 C_3 之间的距离 $DCS_{center}(C_1,C_3)$=6,C_2 和 C_3 之间的距离 $DCS_{center}(C_2,C_3)$=3.

未指定类型的实体 s_5 所在的簇 C_3 与电影实体所在的簇 C_2 之间的距离更近,因此将 C_3 合并到 C_2 中,即将实体 s_5 根据所设计的存储方案存储到类型为“电影”的顶点表中.

定义 5(最优实体簇集合). 实体簇集合 C 满足:(1) 集合中的所有实体簇都包含具有明确类型的实体,且两个实体簇不包含相同类型的实体;(2) 集合中的所有实体的最近距离实体都在其所在的实体簇中.

下面给出算法 1 的正确性证明及复杂度证明.

定理 1. 给定实体集合 S,算法 1 能够给出最优的实体簇集合 C.

证明:算法 1 首先将根据数据集中数据本身的特点进行数据类型归类:若实体 s 为有类型数据,则将其归入实体 s 对应类型 τ 的实体簇 C_τ 中,并将 C_τ 合并到实体簇集合 C 中;若实体 s 为无类型数据,则将其分别归入一个单独的用于处理无类型数据的实体簇 C_0 中,在这一步中,能够确保定义 5 中的第 1 条要求,经过第一轮迭代,各个实体都将归并到某个实体簇中.在后续的迭代过程中,每轮迭代都将会两两计算实体簇 C_1 和 C_2 之间的距离 $DCS_{cluster}(C_1,C_2)$,并在所有的距离中找到距离最小的两个实体簇 C_i 和 C_j,其中,要求 $i\neq j$.若找得到满足条件的两个实体簇,则进行这两个实体簇的合并;否则聚类过程结束.在首轮迭代之后的迭代,确保了定义 5 中的第 2 条要求.最终找到最优的实体簇集合 C.证毕. □

算法 1 的时间复杂度由两部分组成:(1) 算法经过 e 次迭代;(2) 在每次迭代中,需要两两对比实体簇距离,在最坏情况下,每个实体都是单独的实体簇,此时两两计算实体簇距离的复杂度为 $O(s^2)$.因此,算法 1 的时间复杂度为 $O(es^2)$.

4 查询语言互操作

SPARQL 是 RDF 图上的查询语言,Cypher 是属性图之上的主要查询语言.两种不同的查询语言在语法上有较大差异,KGDB 以第 3 节所介绍的统一的存储方案为基础,建立在存储方案之上的查询过程,可以使用两种不同的查询语言查询同一个知识图谱,从而达到查询语言互操作的目的.在文献[35,36]给出了 RDF 和 Cypher 的形式语义,KGDB 系统将 SPARQL 和 Cypher 语言看作“统一查询语义”的两种不同“语法视图”,关系模型被作为物理存储模型,将 RDF 图和属性图进行统一存储,同时对齐 SPARQL 语言和 Cypher 语言的查询语义.

知识图谱查询处理的最基本算子即是基本图模式匹配查询(basic graph pattern,简称 BGP),这类查询可以看作子图匹配或者子图同构查询.已有的各种知识图谱查询语言均以子图匹配作为核心算子.虽然图数据上的子图匹配查询算法已有很多,但却缺少同时具备系统性和高效性的面向大规模知识图谱的子图匹配查询处理算法.KGDB 基于 SPARQL 和 Cypher 语言处理子图匹配查询,需要建立起两种语言的联系,将同样的查询意图转化为不同的两种查询表达式,同时保障两种查询语言处理结果的正确性与高效性.

在本节中将会用到关系代数中的几个经典算子,如 ρ(重命名)、π(投影)、σ(选择)、$\bowtie$ (连接)、×(笛卡尔积)、$\cap$(交)和$\cup$(并)等.使用连接列表 $\mathbb{L}$ 表示抽象语义,$\mathbb{L}=\{r_1,r_2,\ldots,r_n\}$,其中,$r_1,\ldots,r_n$ 表示连接列表中的 n 个关系表.对关系表 r,$r\rightarrow property$ 表示关系表中所有元组属性 $property$ 的值.

4.1 SPARQL查询处理

首先,给出 RDF 图基本图模式匹配查询的形式化定义.

定义 6(RDF 图基本图模式匹配). RDF 图 G 上的基本图模式查询 Q 的语义定义为:

(1) μ是从 $V(Q)$中的顶点到 V 中顶点的映射;

(2) $(G,\mu)\models Q$ 当且仅当对任意$(s_i,p_i,o_i)\in Q$ 满足:i) s_i 和 o_i 可以分别匹配$\mu(s_i)$和$\mu(o_i)$;ii) $(\mu(s_i),\mu(o_i))\in E$;iii) $lab(\mu(s_i),\mu(o_i))=p_i$;

(3) $\Omega(Q)$是使$(G,\mu)\models Q$满足的μ的集合,也就是图 G 上基本图模式查询 G_Q 的答案集合.

例 6:图 5 是一个 RDF 基本图模式匹配查询,查询 Q 中共包含两条三元组 t_1=(Beethoven,composes,?music)和 t_2=(Beethoven,birthDate,“1770-12-16”).两条三元组构成一个简单的星型结构,它所要查询的是 Beethoven 创作的所有作品.在图 2 的 RDF 图中执行这个查询,?music 匹配到 Fate Symphony 上,作为一个变量,可以在查询的结果子句中要求将其输出.

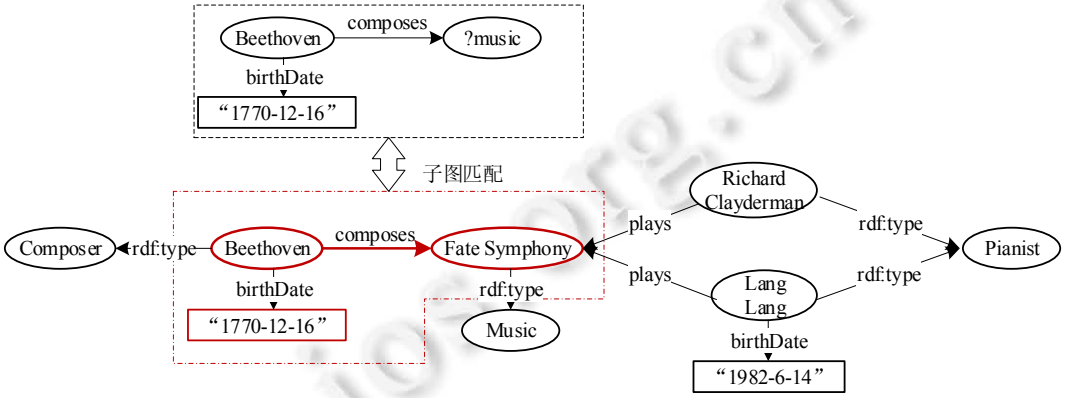

Fig.5 A basic graph pattern matching query example

图 5 基本图模式查询示例

最简单的 SPARQL 查询语句包括两个部分:SELECT 关键字引导的结果子句和 WHERE 关键字引导的约束子句.KGDB 根据 SPARQL 查询语句的语法进行语义解析,生成语义树.根据 SPARQL 查询语句中每部分的语义,使用下面的转换规则自底向上构建关系代数形式的查询语义树.

规则 6. 对于出现在 SPARQL 查询约束子句中的任意三元组 $t=(s,p,o)$,若$\varphi(t_i)=mem$,则在语义树的底层添加一个叶子结点 $r_s=\rho_s(r)$,其中,$name(r)=o$.即,把关系 r_s 添加到连接列表$\mathbb{L}$中.

规则 7. 对于出现在 SPARQL 查询约束子句中的任意三元组 $t=(s,p,o)$,若$\varphi(t)=prop$,已知 $r_s\in\mathbb{L}$,则将$\mathbb{L}$中的 r_s 更改为原 r_s 与属性约束$\sigma_{r\to p=o}(r)$(其中,$rel(s)=r$)的交.

规则 8. 对于出现在 SPARQL 查询约束子句中的任意三元组 $t=(s,p,o)$,若$\varphi(t)=edge$,若 $r_s\in\mathbb{L}$,则将$\mathbb{L}$中的 r_s 更改为施加连接约束 $r_s \bowtie|_{r_s.id=r_p.start} r_p$ 后的 r_s;若 $r_o\in\mathbb{L}$,则将$\mathbb{L}$中的 r_o 更改为施加连接约束 $r_o \bowtie|_{r_o.id=r_p.end} r_p$ 后的 r_o.对于主语(宾语)为 URI 的情况,对谓语对应的关系表 r_p 施加选择约束 $\sigma_{r_p.start=s}(r_p)(\sigma_{r_p.end=s}(r_p))$.

规则 9. 对于出现在 SPARQL 查询结果语句中的任意变量 var,添加投影约束:$\pi_{var.id,var.property}(r_{final})$,其中,$r_{final}$为连接列表$\mathbb{L}$中的所有关系进行笛卡尔积后得到的最终关系结果.

对应的,SPARQL 语句到查询语义的转化算法如下.

算法 2. SPARQL 基本图模式匹配查询处理.

输入:查询中的三元组集合 $T=\{t_1,t_2,...,t_{n_1}\}$,其中,$t_i=(s_i,p_i,o_i)$;查询中的变量集合 $VAR=\{var_1,var_2,...,var_{n_2}\}$;关系表集合 $R=\{r_1,r_2,...,r_{n_3}\}$;

输出:图模式匹配查询的结果关系 r_{result}.

1. $\mathbb{L}=\varnothing$;
2. $r_{result}=\varnothing$;
3. **for each** $t_i\in T$ **do**
4. **if** $\varphi(t_i)=mem$ **then** //将名为 o_i 的关系表重命名后加入到连接操作列表
5. $\mathbb{L}\leftarrow\mathbb{L}\cup\{r_{s_i}\mid r_{s_i}=\rho_{s_i}(r)\wedge r\in R\wedge name(r)=o_i\}$;

6. **else if** $\varphi(t_i)=prop$ **then** //添加属性约束
7. $\mathbb{L} \leftarrow \mathbb{L} \setminus \{r_{s_i}\}$;
8. $r_{s_i} \leftarrow r_{s_i} \cap \sigma_{r \to p_i=o_i}(r)$,其中,$r \in R \wedge rel(s_i)=r$;
9. $\mathbb{L} \leftarrow \mathbb{L} \cup \{r_{s_i}\}$;
10. **else** //添加连接约束
11. **if** $s_i \in U, o_i$ 为变量 **then** //主语为常量,宾语为变量
12. $\mathbb{L} \leftarrow \mathbb{L} \setminus \{r_{o_i}\}$;
13. $r_{o_i} \leftarrow r_{o_i} \bowtie_{r_{o_i}.id=r_p.start} (\sigma_{r_p.start=s_i}(r_p))$,其中,$r_p \in R \wedge name(r_p)=p_i$;
14. $\mathbb{L} \leftarrow \mathbb{L} \cup \{r_{o_i}\}$
15. **else if** $o_i \in U_i, s_i$ 为变量 **then** //宾语为常量,主语为变量
16. $\mathbb{L} \leftarrow \mathbb{L} \setminus \{r_{s_i}\}$;
17. $r_{s_i} \leftarrow r_{s_i} \bowtie_{r_{s_i}.id=r_p.start} (\sigma_{r_p.end=o_i}(r_p))$,其中,$r_p \in R \wedge name(r_p)=p_i$;
18. $\mathbb{L} \leftarrow \mathbb{L} \cup \{r_{s_i}\}$
19. **else** //主语和宾语均为变量
20. $\mathbb{L} \leftarrow \mathbb{L} \setminus \{r_{s_i}, r_{o_i}\}$;
21. $r_{o_i} \leftarrow r_{o_i} \bowtie_{r_{o_i}.id=r_p.start} r_p$,其中,$r_p \in R \wedge name(r_p)=p_i$;
22. $r_{s_i} \leftarrow r_{s_i} \bowtie_{r_{s_i}.id=r_p.start} r_p$,其中,$r_p \in R \wedge name(r_p)=p_i$;
23. $\mathbb{L} \leftarrow \mathbb{L} \cup \{r_{s_i}, r_{o_i}\}$;
24. **end**
25. **for each** $var_i \in Var$ **do** //添加投影约束,将对应关系表中的信息输出
26. $r_{result} \leftarrow r_{result} \cup \pi_{var_i.id, var_i.property}(r_{final})$; //$r_{final}$ 为$\mathbb{L}$中关系表做笛卡尔积得到的结果
27. **end**

算法 2 遍历 SPARQL 查询中涉及的三元组,针对不同类型的三元组做出不同的应对措施,最终形成关系代数表示的查询语义.与存储方案相似,查询处理也将三元组 $t_i=(s_i,p_i,o_i)$分为 3 种类型:表明实体类型的$\varphi(t_i)=mem$、宾语部分为字面量的$\varphi(t_i)=prop$ 和其他形式($\varphi(t_i)=edge$)的三元组.当处理 *mem* 类型的三元组时(第 4 行、第 5 行),在连接列表中添加表名为该条三元组宾语 o_i 的关系表,并将这个关系表重命名为该条三元组的主语 s_i(在 SPARQL 查询中,这种 *mem* 类型的三元组的主语部分通常是变量,且在同一条查询中的其他的三元组中使用相同的变量代称);当处理 *prop* 类型的三元组时(第 6 行~第 9 行),向约束集合中添加一条属性约束;当处理其他类型的三元组(第 10 行~第 24 行),即 *edge* 类型的三元组时,添加连接约束.第 25 行、第 26 行处理所有投影约束,将用户要求的查询结果进行最终输出.

定理 2. 给定 SPARQL 查询中三元组集合 T 和关系表集合 R,算法 2 输出的结果是正确的.

证明:算法 2 遍历 SPARQL 查询中涉及的所有三元组,并根据每条三元组 $t_i=(s_i,p_i,o_i) \in T$ 的三元组类别,给出对应的方案.参见定义 3 可知,三元组的语义类型仅有 3 种,即算法 2 对于所有语义的三元组,均可以找到对应的解决方案,即给出正确的关系表的连接列表$\mathbb{L}$.另一方面,结果子句中出现的所有变量,都在约束子句中出现,添加投影约束仅改变最终输出结果,不影响算法的正确性.证毕. □

算法 2 的时间复杂度由两部分组成:(1)为了生成关系表的连接列表$\mathbb{L}$,算法 2 需要遍历 SPARQL 查询中所有三元组,并对应给出解决方案,其时间复杂度为 $O(n)$;(2)向关系表的连接列表$\mathbb{L}$添加新的条目的时间复杂度是常数的,即 $O(k)$.因此,算法 2 的时间复杂度为 $O(kn)$.

4.2 Cypher查询处理

与 SPARQL 查询处理类似,首先给出属性图上的模式匹配查询的形式化定义.

定义 7(属性图模式). $\alpha=(a,Lab,Map)$为一个点模式,其中:$a\in\mathcal{A}\cup\{nil\}$为点模式可选的名字,$\mathcal{A}$为名字的集合,*nil* 为空;*Lab* 为可空的点的有限标签集合,且 $Lab\subseteq\mathcal{L}$,$\mathcal{L}$为属性图的标签集合;*Map* 为可空的属性集合$\mathcal{K}$到属性值 *Val* 的映射.$\beta=(d,Lab,a,Map)$为一个边模式,其中,$d\in\{\leftarrow,\rightarrow\}$表示边模式的方向;*Lab* 为可空的边的有限标签集合,且 $Lab\subseteq\mathcal{L}$;$a\in\mathcal{A}\cup\{nil\}$为边模式可选的名字;*Map* 为可空的属性集合$\mathcal{K}$到属性值 *Val* 的映射.$\omega=\alpha_1\beta_1\alpha_2\beta_2\ldots\beta_{n-1}\alpha_n$为一个路径模式,其中,$\alpha_i$为点模式,$\beta_i$为边模式.

定义 8(属性图模式匹配). 属性图上的图模式匹配可以递归定义如下[36].

(1) 对点模式$\alpha=(a,Lab,Map)$,其在属性图 G 上的模式匹配为$(v,G,\mu)\vDash\alpha$,则满足 a 为 *nil* 或者$\mu(a)=v$,$Lab\subseteq\lambda(v)$且$[[\gamma(v,k)=Map(k)]]_{G,\mu}$成立;

(2) 对于长度为 $m=0$,即只包含一个点的路径$\mathcal{P}$,在属性图 G 上的模式匹配为$(v\cdot\mathcal{P},G,\mu)\vDash\alpha\beta\omega$,则满足:

a) a 为 *nil* 或者$\mu(a)=list(\cdot)$;

b) $(v,G,\mu)\vDash\alpha$并且$(\mathcal{P},G,\mu)\vDash\omega$;

(3) 对于长度为 $m\geqslant1$ 的路径$\mathcal{P}$,在属性图 G 上的模式匹配为$(v_1e_1v_2\ldots e_mv_{m+1}\cdot\mathcal{P},G,\mu)\vDash\alpha\beta\omega$,则满足下列条件:

a) a 为 *nil* 或者$\mu(a)=list(e_1,\ldots,e_m)$;

b) $(v_1,G,\mu)\vDash\alpha$并且$(\mathcal{P},G,\mu)\vDash\omega$;

c) $Lab_\beta\subseteq\{\lambda(e_1)\cup\lambda(e_2)\cup\ldots\cup\lambda(e_m)\}$;

d) $[[\gamma(e_i,k)=Map(k)]]_{G,\mu}$成立;

e) $(src(e_i),tgt(e_i))=\begin{cases}\{(v_i,v_{i+1})\}, & \text{当}d\text{为}\rightarrow\\ \{(v_{i+1},v_i)\}, & \text{当}d\text{为}\rightarrow\end{cases}$.

则对于 Cypher 查询 Q,其结果集为

$$match(Q,G,\mu)=\biguplus_{\mathcal{P}\text{ in }G}\{\omega\mid(\mathcal{P},G,\mu)\vDash Q\}\tag{5}$$

其中,$\uplus$表示包的合并.

与 SPARQL 查询语句相似,最简单的 Cypher 查询也主要包括两个部分:MATCH 关键字引导的约束子句和 RETURN 关键字引导的结果子句.KGDB 使用下列转换规则对 Cypher 语句进行处理.

规则 10. 对于出现在 Cypher 查询语句中的所有点模式$\alpha=(a,Lab,Map)$,向关系表的连接列表$\mathbb{L}$中添加 n 个关系表 $r_1,\ldots,r_n$,其中,$Lab\subseteq\{rel(r_1),rel(r_2),\ldots,rel(r_n)\}$,并施加属性约束 $\sigma_{r_i\rightarrow domain(Map)=range(Map)}(r_i),i\in[1,n]$,其中,*domain*(*Map*)指 *Map* 的定义域,*range*(*Map*)指 *Map* 的值域.

规则 11. 对于出现在 Cypher 查询语句中的所有边模式$\beta=(d,Lab,a,Map)$,向$\mathbb{L}$中关系表施加连接约束:

$$\begin{cases}r_{v_{i+1}}\bowtie_{r_{v_{i+1}}.id=r_e.start}r_e\ \text{AND}\ r_{v_i}\bowtie_{r_{v_i}.id=r_e.end}r_ed=\leftarrow\wedge(src(e),tgt(e))=(v_{i+1},v_i)\wedge Lab\subseteq\lambda(e)\\ r_{v_i}\bowtie_{r_{v_i}.id=r_e.start}r_e\ \text{AND}\ r_{v_{i+1}}\bowtie_{r_{v_{i+1}}.id=r_e.end}r_ed=\rightarrow\wedge(src(e),tgt(e))=(v_i,v_{i+1})\wedge Lab\subseteq\lambda(e)\end{cases}.$$

规则 12.对于出现在 Cypher 查询语句 RETURN 子句中所有的变量 *var*,添加投影约束:$\pi_{var.id,var.property}(r_{final})$,其中,$r_{final}$为连接列表$\mathbb{L}$中的所有关系进行笛卡尔积后得到的最终关系结果.

可以看出:对照 KGDB 统一的存储模型,Cypher 语义的转化相较 SPARQL 更加容易,SPARQL 需要进行三元组的分类,根据具体分类来进行关系代数的映射;而 Cypher 直接根据查询中每一部分的所属集合即可确定语义.

定理 3. 运用上述规则 10 到规则 12,可以正确地将 Cypher 查询语句转化为关系代数表示的查询语义.

证明:基本的 Cypher 语句中的 MATCH 子句的转化在规则 10 和规则 11 中完成,RETURN 子句的转化在规则 12 中定义.规则 10 和规则 11 分别针对 MATCH 子句中的点模式和边模式进行约束转换:(1) 当遇到带标签的点模式$\alpha=(a,Lab,Map)$时,向关系表的连接列表$\mathbb{L}$中添加 n 个关系表 $r_1,\ldots,r_n$,其中,$Lab\subseteq\{rel(r_1),rel(r_2),\ldots,rel(r_n)\}$,并施加属性约束 $\sigma_{r_i\rightarrow domain(Map)=range(Map)}(r_i),i\in[1,n]$,其中,*domain*(*Map*)指 *Map* 的定义域,*range*(*Map*)指 *Map*

的值域;(2) 当遇到带标签的边模式β=(d,Lab,a,Map)时,添加两条连接约束.规则 12 中出现的所有变量都须在 MATCH 子句中出现过,并在执行计划中对应添加投影约束$\pi_{var.id,var.property}(r_{final})$,其中,$r_{final}$为连接列表$\mathbb{L}$中的所有关系进行笛卡尔积后得到的最终关系结果.对于两个子句中可能出现的所有模式匹配内容,都可以找到对应的查询解决方案.证毕. □

例 7:图 6 是一个属性图模式匹配查询,它查询的是音乐家 Beethoven 创作的所有作品.在图 3 的属性图中执行这个查询,可以得到虚线框标记的结果.

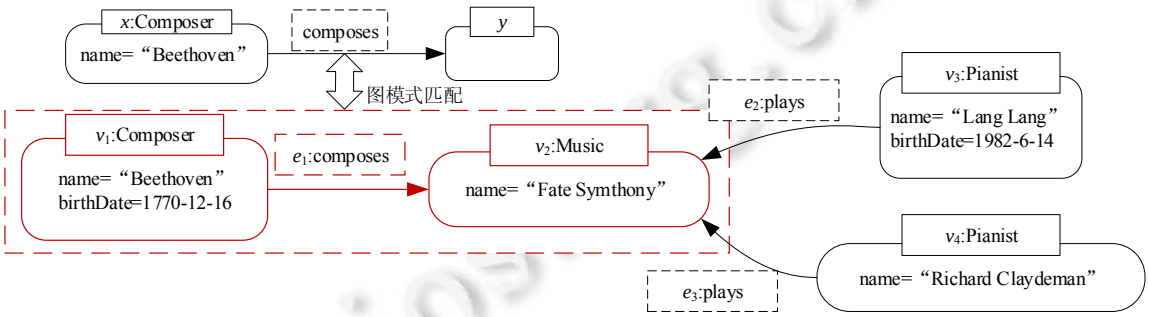

Fig.6 A basic pattern matching query example for Cypher

图 6 Cypher 图模式匹配查询举例

4.3 查询语言语义对齐

图 7 展示了将 SPARQL 查询语句与 Cypher 查询语句进行语义对齐的过程.两种查询语言可以使用完全不同的语法表达相同的语义,相同的查询意图可以被分别表示为两种不同的查询语言.两种查询语言应用各自的转换规则,可以转换成相同的关系代数表示的查询语义,为后面的查询执行提供了统一的语义.

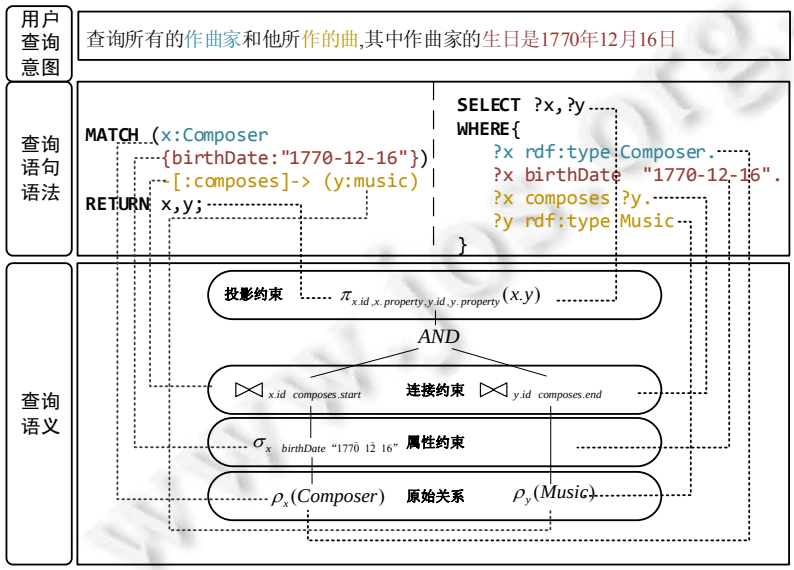

Fig.7 Semantic alignment

图 7 语义对齐

在统一的存储模型之上提供两种查询语言接口,可以在解决具体问题时为使用者提供更多选择,进行两种查询语言的语义对齐,实际上是对两种语言的扩展.

例 8:对于图 7 中的查询语句举例,下面给出两种语言分别的转化过程.

(1) SPARQL 查询转化

运用第 4.1 节介绍的规则,转化 SPARQL 查询语句的过程如下.

a) 对三元组 t_1=(?*x*,*rdf*:*type*,*Composer*)和 t_4=(?*y*,*rdf*:*type*,*Music*),运用规则 6,可知$\varphi(t_1)=\varphi(t_4)=mem$,则将对应的关系表 Composer 和 Music 加入到连接列表中,并将其重命名$\rho_x(Composer)$,$\rho_y(Music)$;

b) 对三元组 t_2=(?*x*,*birthDate*,“1770-12-16”),运用规则 7,可知$\varphi(t_2)=prop$,则向重命名后的表 *x* 添加约束条件$\sigma_{x\to birthDate=\text{“1770-12-16”}}(x)$;

c) 对三元组 t_3=(?*x*,*composes*,?*y*),运用规则 8,可知$\varphi(t_3)=edge$,则添加关系表之间的连接约束:

$$r_x \bowtie_{r_x.id=r_{composes}.start} r_{composes}, r_y \bowtie_{r_y.id=r_{composes}.end} r_{composes};$$

d) 对结果子句中的所有变量,即 *x* 和 *y*,运用规则 9,添加投影约束$\pi_{x.id,x.property,y.id,y.property}(r_{final})$,其中,$r_{final}$ 为对连接列表中所有关系表施加相应约束后做笛卡尔积的关系表.

至此,该条 SPARQL 语句成功转换为图 7 中的语义树.

(2) Cypher 查询转化

运用第 4.2 节介绍的规则,转化 Cypher 查询语句的过程如下.

a) 对点模式α=($\{x,y\}$,$\{Composer,Music\}$,*birthDate*→“1770-12-16”),运用规则 10,向连接列表中添加关系表 Composer 和 Music,并进行相应重命名,将其加入到连接列表$\mathbb{L}$中:$\rho_x(Composer)$和$\rho_y(Music)$.并添加约束条件$\sigma_{x\to birthDate=\text{“1770-12-16”}}(x)$;

b) 对 Cypher 语句中的边模式β=(→,$\{composes\}$,*nil*,$\{\cdot\}$),则根据规则 11,添加相应的连接约束:

$$r_x \bowtie_{r_x.id=r_{composes}.start} r_{composes}, r_y \bowtie_{r_y.id=r_{composes}.end} r_{composes};$$

c) 对 RETURN 子句中的所有变量,即 *x* 和 *y*,根据规则 12,添加投影约束$\pi_{x.id,x.property,y.id,y.property}(r_{final})$,其中,$r_{final}$ 为对连接列表中所有关系表施加相应约束后,做笛卡尔积的关系表.

根据本节介绍的 SPARQL 和 Cypher 查询语言相应的查询语句转化规则,能够将两种查询语言转化为相同的使用关系代数表达的抽象语义树(查询语言的语义对齐方法的正确性分析在定理 2 和定理 3 中分别给出),使得 KGDB 在兼容两种语法完全不同的查询语言的前提下统一了查询的语义.这为查询处理后面的优化过程提供了便利,并为用户在同一个知识图谱上的查询提供了更多选择.

5 实 验

本节在合成数据集和真实数据集上验证统一存储方案的高效性和和查询语言的互操作性,并且与 RDF 三元组库 gStore[2]和属性图数据库 Neo4j[3]进行比较.

5.1 实验环境和数据集

本文使用的实验平台为单节点服务器,节点配置主频为 2.5GHz 的 Intel(R) Xeon(R) Platinum 8255C 八核处理器,其内存大小为 32GB,硬盘大小为 100GB,使用 Linux 64-bit CentOS 7.6 操作系统.

KGDB 以开源属性图数据库 AgensGraph 为后端.使用的 Neo4j 版本为 4.1.0 社区版,使用 neosemantics 插件 4.0.0.1 版本,以在 Neo4j 中支持 RDF 图的存储,使用的 gStore 版本为 0.7.2.本文进行三个系统的存储效率的对比实验,并在 KGDB 和 gStore 系统上进行 SPARQL 基本图模式查询对比实验,在 KGDB 和 Neo4j 上进行 Cypher 查询对比实验.从存储空间、存储时间和查询效率上进行系统的全面评估.

本文使用的数据集包括 LUBM[37]标准合成数据集以及 DBpedia[38]真实数据集.LUBM 允许用户定义数据集的大小,本文使用了五个不同规模的 LUBM 数据集(即 LUBM10,LUBM20,LUBM30,LUBM40 和 LUBM50)进行实验测试和比较;DBpedia 数据集是从维基百科中提取实体关系生成的一个真实数据集.本次实验中,所有数据集均以 *N*-Triple 格式表示,表 2 给出了实验数据集的具体统计信息.

Table 2 Datasets

表 2 数据集

数据集	元组数	顶点个数	边个数	文件大小
LUBM10	1 316 700	207 429	630 757	208M
LUBM20	2 782 126	437 558	1 332 030	441M
LUBM30	4 109 002	645 957	1 967 309	651M
LUBM40	5 495 742	864 225	2 630 657	871M
LUBM50	6 890 640	1 082 821	3 298 814	1.1G
DBpedia	23 445 441	2 257 499	6 876 041	3.1G

在 LUBM 数据集上执行的查询来自 LUBM 提供的 14 个标准查询中的 8 个(即 Q1~Q6,Q11 和 Q14).其中,

- Q1~Q3 和 Q14 为不涉及推理的 SPARQL 查询:(1) Q1 输入数据量大,选择度高;(2) Q2 为涉及 3 个实体的三角查询;(3) Q3 查询涉及的类型数据量大;(4) Q14 输入数据量大,选择度低;
- Q4~Q6 和 Q11 为涉及到推理的查询.

gStore 并不支持 RDF 图上的推理功能,而 Neo4j 也仅仅可以需要通过插件的方式支持推理功能.同样的,KGDB 也尚未支持推理查询,故本文仅在 gStore 和 KGDB 上进行推理查询返回空结果的查询效率对比实验.LUBM 数据集中涉及推理的查询可以简单分为 4 类:(1)Q4~Q9 涉及到 subClassOf 层次类型;(2)Q5 包含 subPropertyOf 层次关系,查询中包含的内容需要借助本体信息才可直接执行;(3)Q6~Q10 需要进行层次类型推理,即查询中涉及的实体层次类型关系在本体信息中也未直接给出;(4)Q11~Q13 需要进行更加复杂的关系推理,即除了 subClassOf 和 subPropertyOf 关系之外的关系推理.本文在 LUBM 数据集上的实验部分在 4 类推理查询中各选择一个进行比较实验.因为缺乏 DBpedia 数据集上的查询模板,本文设计了 4 个包含不同数据量的查询(记为 Q_dbp1~Q_dbp4).Q_dbp2~Q_dbp4 的基本结构相同,不同的只是数据量的大小:Q_dbp4 数据量最大,达到数百万条;而 Q_dbp2 最小.

5.2 实验结果

本节在 3 个方面进行实验结果比较,包括存储时间、存储空间和 SPARQL 或 Cypher 的查询效率.以下实验结果中,每个查询测试均进行 3 次取平均值,避免随机误差.

5.2.1 存储时间

如图 8(a)所示:在存储时间上,KGDB 比 gStore 和 Neo4j 需要的时间短,gStore 所需的存储时间最长,并且随着数据量的增长,各个系统之间的时间差异越来越大,KGDB 的优势越加明显;在数据量最大的数据集 DBpedia 上,KGDB 可以达到 gStore 及 Neo4j 存储速度的将近 10 倍,将存储效率提高一个数量级.

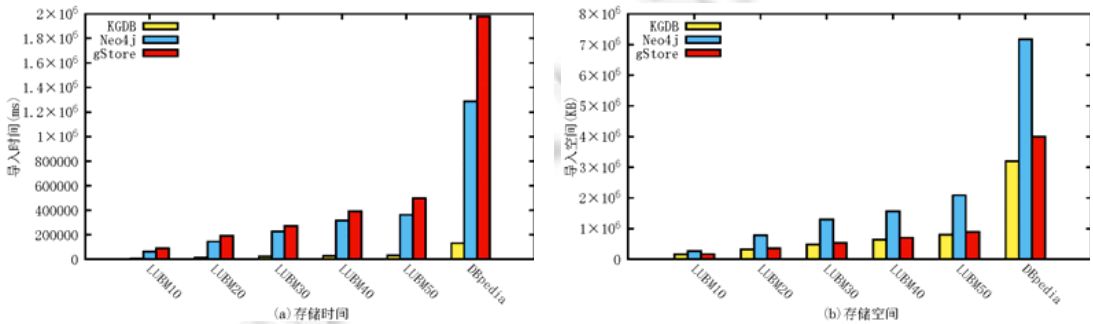

Fig.8 Results for data insertion and storage space

图 8 存储时间、存储空间实验结果

在存储处理过程中,KGDB 仅需要一次无类型实体聚类过程,即可多次进行数据集的存储,在存储过程中,不需要复杂的转换过程.对应的,gStore 需要进行字符串与 id 的转换、VS 树的建立等等过程,Neo4j 需要进行类型到标签的转化等等过程.

5.2.2 存储空间

如图 8(b)所示:在存储空间上,KGDB 也相较 gStore 和 Neo4j 优势明显.3 个系统中:Neo4j 所需存储空间大于数据集本身的大小,最高可以达到数据集本身的 2 倍;gStore 可以将数据集压缩存储,最大压缩率达到 0.8;相比之下,KGDB 最大压缩率达到 0.7,实现数据的高效存储.随着数据量的增长,使用字典编码的 KGDB 在存储空间上的优势越加明显.需要注意的是:Neo4j 社区版仅可以使用两个数据库(database),而每个数据库仅能配置一个图(graph),这就要求用户在需要建立多个知识图谱数据库时,做 Neo4j 系统的全备份.虽然专业版提供了多数据库的支持,但社区版的配置无疑限制了普通用户的使用,也增加了存储多个独立知识图谱的存储空间要求.

在本文的实验中,仅计算单个数据库在装载知识图谱前后的空间变化.如果将系统空间要求计算在内,Neo4j 的知识图谱存储所需空间会更大.

5.2.3 查询效率

查询效率实验分别在 LUBM 合成数据集和 DBpedia 真实数据集上进行.在 LUBM 数据集上,采用了 4 个基本查询和 4 个涉及推理的查询.在 DBpedia 上,设计了 4 个查询,其中:Q_dbp1 输入数据量大,选择度高;Q_dbp2~Q_dbp4 具有相同的结构,数据量依次增大.在同样的语义下,分别构造 SPARQL 和 Cypher 查询语句,并在对应系统上进行实验.

(1) SPARQL 查询效率实验.

gStore 是 RDF 图数据库,使用 SPARQL 作为其查询语言,可以支持大规模 RDF 知识图谱的数据管理任务.KGDB 与 gStore 系统的 SPARQL 查询效率对比实验结果如图 9 所示:gStore 不能支持最复杂的涉及 3 个实体的三角查询 Q2,而 KGDB 可以在较短时间内完成这一查询.对于 Q3,随着数据量的增加,KGDB 与 gStore 相比,其查询时间增长幅度更小,表明 KGDB 在大规模知识图谱数据上的查询效率优于 gStore.在其他两个查询 Q1 和 Q14 上,KGDB 可以达到与 gStore 相同的数量级,因此具有可比性.

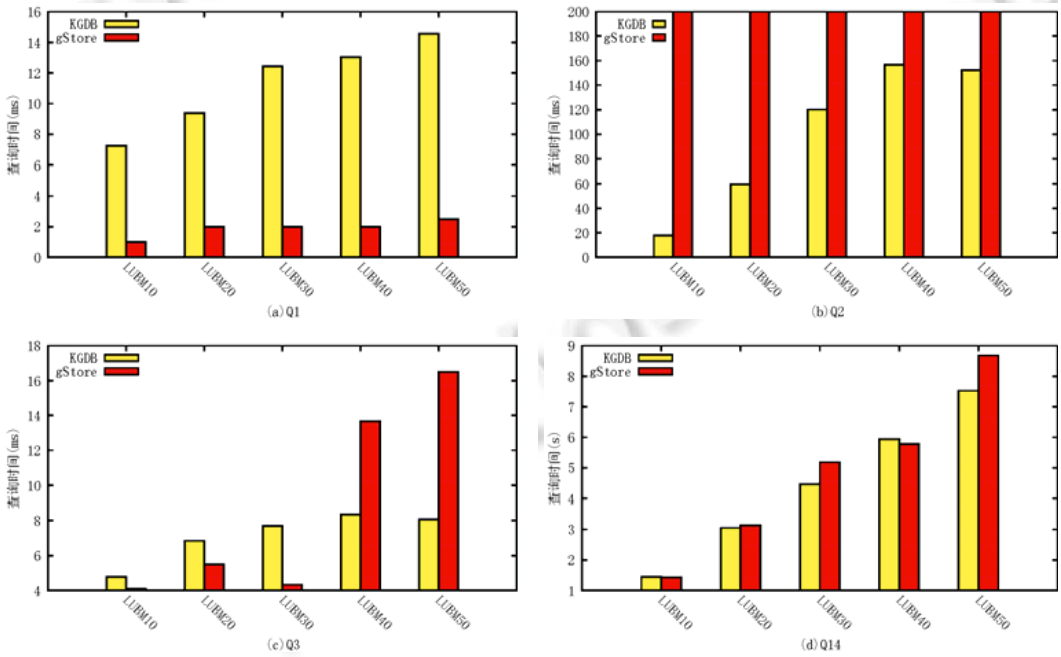

Fig.9 Query efficiency experimental results on LUBM by SPARQL for KGDB and gStore

图 9 在 LUBM 数据集上的 KGDB 和 gStore 的 SPARQL 查询效率实验结果

LUBM 提供的标准查询 Q2 是选取的 4 个不涉及推理的查询中最为复杂的,在 gStore 系统上将会引起系统错误,不能够直接执行.为了公平比较,没有进行查询的改写.对于查询 Q1 和 Q3,其基本结构是一致的,区别在于

涉及的实体的数据量不同,Q3 的输入数据量更大.而相较而言,KGDB 可以在 Q3 上呈现出平缓的查询时间增长趋势,说明 KGDB 可以应对大规模知识图谱上的查询,数据量的增长不会导致查询效率的降低.对于查询时间最长的 Q14,该查询涉及的实体数量巨大,在 LUBM50 上,查询结果将需返回数十万条数据.在这一查询上,KGDB 和 gStore 的查询时间可以达到同一数量级.

如图 10 所示,在 gStore 和 KGDB 上的 SPARQL 推理查询实验结果表明:两个系统均不支持涉及推理的查询,但都能在较短时间内给出查询结果,虽然因缺乏推理功能导致查询的最终结果为空.对于相同的 LUBM 标准查询,KGDB 能够在更短的时间内给出查询结果,并且查询时间随数据集规模增长的幅度与 gStore 相比更小.

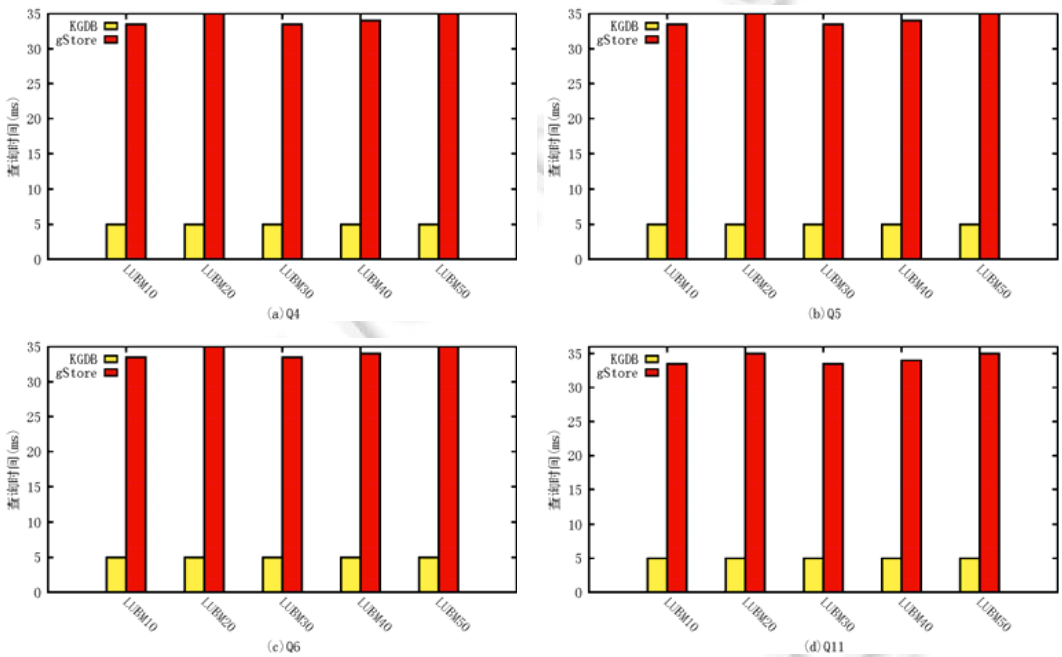

Fig.10 Inference query efficiency experimental results on LUBM by SPARQL for KGDB and gStore

图 10 在 LUBM 数据集上的 KGDB 和 gStore 的 SPARQL 推理查询效率实验结果

如图 11 所示,可以得出结论:在 DBpedia 数据集上,对于查询 Q_dbp1~Q_dbp3,KGDB 可以达到比 gStore 更快的查询速度,KGDB 在最优的查询(Q_dbp1)上,可以将查询速度提升一个数量级.这说明在选择算子的处理上,KGDB 拥有较大优势.而对于最慢的查询(Q_dbp4),KGDB 也可以达到与 gStore 相同的数量级.

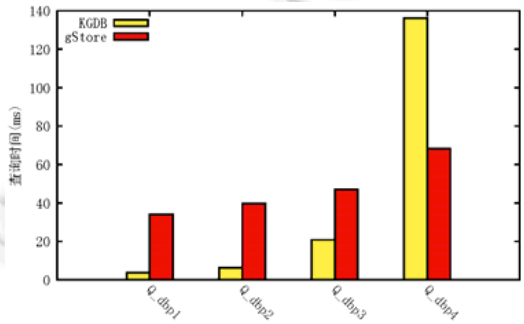

Fig.11 Query efficiency experimental results on DBpedia by SPARQL for KGDB and gStore

图 11 在 DBpedia 数据集上的 KGDB 和 gStore 的 SPARQL 查询效率实验结果

(2) Cypher 查询效率实验.

Neo4j 是属性图数据库,使用 Cypher 作为查询语言,支持完整的 ACID 规则,可以更加快速地处理连接数据.下面将进行 KGDB 和 Neo4j 系统的 Cypher 查询效率对比实验.

如图 12 所示:在 LUBM 数据集上,KGDB 在 3 个标准查询 Q1,Q3 和 Q14 上都可以达到比 Neo4j 更快的速度;在最优的查询上(Q3),KGDB 比 Neo4j 快近 70 倍;而在最慢的查询上(Q2),KGDB 也可以达到与 Neo4j 相同的数量级.

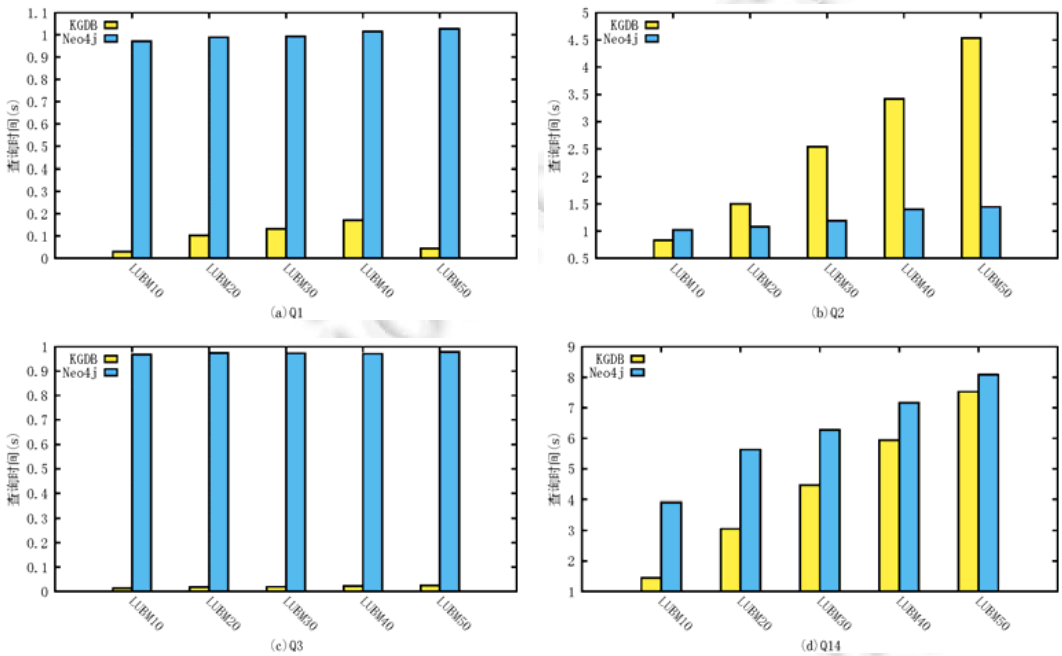

Fig.12 Query efficiency experimental results on LUBM by SPARQL for KGDB and Neo4j

图 12 在 LUBM 数据集上的 KGDB 和 Neo4j 的 SPARQL 查询效率实验结果

对于最复杂的三角查询 Q2,KGDB 的查询速度慢于 Neo4j.这是因为 Neo4j 的存储方式使之查询结点之间的连接变得容易,而不需要像 KGDB 这种关系数据库,执行耗时的连接(JOIN)操作.但是,KGDB 相较于 Neo4j 拥有更多优势:(1)KGDB 不需要单独的插件,原生的支持 RDF 图和属性图的统一存储,并在存储过程中,比 Neo4j 需要更短的存储时间和更小的存储空间,同时管理多个知识图谱更加容易;(2)KGDB 同时支持 SPARQL 和 Cypher 查询语言,允许两个查询语言在同一知识图谱上的互操作.

在 DBpedia 数据集上的实验结果表明(如图 13 所示):在所有查询上,KGDB 都可以达到比 Neo4j 更快的查询速度.对于最优的查询(Q_dbp3),KGDB 可以比 Neo4j 快 2 个数量级;而最慢的查询(Q_dbp4)上,KGDB 也可以达到 14 倍于 Neo4j 的查询速度.

从 LUBM 上的实验结果随数据量增长呈现的趋势上来看:KGDB 在数据量不断增长后,相较 Neo4j 的优势逐渐变小,但查询时间仍然低于 Neo4j.在最复杂的查询上(Q2)也可以得出相似的结论,即:随着需要遍历的数据量增大,Neo4j 查询时间的增长幅度相较 KGDB 小,但这种增长不会带来数量级的变化.而在 DBpedia 上的实验结果表明:因为真实数据集所表现出的半结构化和稀疏性,查询涉及数据量的增长将会为 KGDB 带来优势.

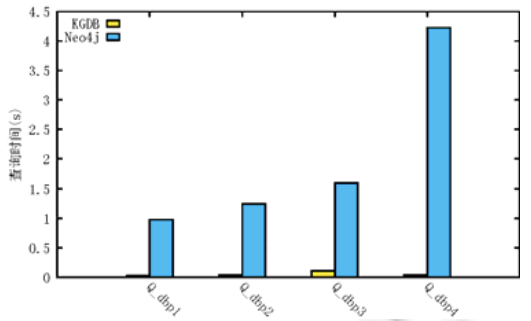

Fig.13 Query efficiency experimental results on DBpedia by Cypher for KGDB and Neo4j

图 13 在 DBpedia 数据集上的 KGDB 和 Neo4j 的 Cypher 查询效率实验结果

6 总 结

本文研发了统一模型和语言的知识图谱数据库管理系统 KGDB.

(1) 统一存储:支持存储 RDF 图和属性图数据,使用字典编码,提高存储效率,节省存储空间,使用基于特征集的聚类方法,解决无类型实体的存储问题,使得具有相近语义的实体存储在同一关系表中,提高查询效率;

(2) 互操作语法层:进行两种数据模型之上的查询语言 SPARQL 和 Cypher 的语义对齐,即对同一知识图谱,使用两种查询语言都可以得到相同的查询结果,达到查询语言互操作的目的;

(3) 统一语义层:SPARQL 和 Cypher 两种查询语言可以通过转换规则,转化为关系代数表示的相同的抽象查询语义树.

真实数据集和合成数据集上的实验验证了 KGDB 系统的存储效率和查询效率,实验结果表明:KGDB 在真实数据集上普遍优于 gStore 和 Neo4j;而在合成数据集上,KGDB 可以与 gStore 和 Neo4j 达到相同数量级.

本文只讨论了单机系统上的知识图谱管理问题,随着知识图谱数据规模的不断增大,分布式知识图谱管理系统将成为研究热点.在后续工作中,我们将会讨论分布式环境下知识图谱的统一存储方案和查询处理方法.

References:

[1] Wang X, Zou L, Wang CK, Peng P, Feng ZY. Research on knowledge graph data management: A survey. Ruan Jian Xue Bao/ Journal of Software, 2019,30(7):2139−2174 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5841.htm [doi: 10.13328/j.cnki.jos.005841]

[2] Zou L, Özsu MT, Chen L. gStore: A graph-based SPARQL query engine. The VLDB Journal, 2014,23(4):565−590.

[3] The Neo4j Team. The Neo4j manual v4.1. 2020. https://neo4j.com/docs/developer-manual/current/

[4] Dgraph Labs, Inc. The Dgraph homepage. 2020. https://dgraph.io/

[5] The HugeGraph Team. The HugeGraph manual. 2020. https://hugegraph.github.io/hugegraph-doc/

[6] Abadi DJ, Marcus A, Madden SR. Scalable semantic Web data management using vertical partitioning. In: Klas W, ed. Proc. of the 33rd Int'l Conf. on Very Large Data Bases. Vienna: VLDB Endowment, 2007. 411−422.

[7] Bornea MA, Dolby J, Kementsietsidis A. Building an efficient RDF store over a relational database. In: Ross K, ed. Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2013. 121−132.

[8] Moerkotte G, Neumann T. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. IEEE Trans.on Data Engineering, 2011,984−994.

[9] Anagnostopoulos I, Mamoulis N, *et al*. Extended characteristic sets: Graph indexing for SPARQL query optimization. In: Proc. of the 2017 IEEE Int'l Conf. on Data Engineering (ICDE). California: IEEE, 2017. 497−508.

[10] Anyanwu K, Kim H, *et al*. Type-Based semantic optimization for scalable RDF graph pattern matching. In: Proc. of the 26th Int'l Conf. on World Wide Web. New York: ACM, 2017. 785−793.

[11] JanusGraph Authors. JanusGraph—Distributed graph database. 2020. http://janusgraph.org/

[12] TigerGraph. TigerGraph—The first native parallel graph. 2020. https://www.tigergraph.com/

[13] Zou L, Peng P. A survey of distributed RDF data management. Journal of Computer Research and Development, 2017,54(6): 1213−1224 (in Chinese with English abstract).

[14] Wang TT, Rong CT, Lu W. Survey on technologies of distributed graph processing systems (in Chinese with English abstract). Ruan Jian Xue Bao/ Journal of Software, 2018,29(3):569−586. http://www.jos.org.cn/1000-9825/5450.htm [doi: 10.13328/j.cnki.jos.005450]

[15] Harris S, Gibbins N. 3store: Efficient bulk RDF storage. In: Volz R, ed. Proc. of the 1st Int'l Workshop on Practical and Scalable Semantic Systems. Sanibel Island: CEUR-WS.org, 2004. 81−95.

[16] Pan Z, Heflin J. DLDB: Extending relational databases to support semantic Web queries. In: Volz R, ed. Proc. of the 1st Int'l Workshop on Practical and Scalable Semantic Systems. Sanibel Island: CEUR-WS.org, 2004. 109−113.

[17] Wilkinson K. Jena property table implementation. In: Smart PR, ed. Proc. of the 2nd Int'l Workshop on Scalable Semantic Web Knowledge Base Systems. Athens, 2006. 35−46.

[18] Abadi DJ, Marcus A, Madden SR. SW-Store: A vertically partitioned DBMS for semantic Web data management. VLDB Journal, 2009,18(2):385−406.

[19] Yuan P, Liu P, Wu B, *et al*. TripleBit: A fast and compact system for large scale RDF data. Proc. of the VLDB Endowment, 2013, 6(7):517−528.

[20] Neumann T, Weikum G. RDF-3X: A RISC-style engine for RDF. Proc. of the VLDB Endowment, 2008,1(1):647−659.

[21] Weiss C, Karras P, Bernstein A. Hexastore: Sextuple indexing for semantic Web data management. Proc. of the VLDBEndowment, 2008,1(1):1008−1019.

[22] Kim H, Ravindra P, *et al*. A semantics-aware storage framework for scalable processing of knowledge graphs on Hadoop. IEEE Trans. on Big Data, 2017:193−202.

[23] Sun W, Fokoue A, Srinivas K. SQLgraph: An efficient relational-based property graph store. In: Sellis T, ed. Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM, 2015. 1887−1901.

[24] The AgensGraph Team. Manual v1.0. 2020. https://bitnine.net/documentations/manual/agens_graph_developer_manual_en.html

[25] Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. O'Reilly Media, Inc., 2013.

[26] Blazegraph by Systap, LLC. Blazegraph. 2020. https://www.blazegraph.com/

[27] OpenLink Software. OpenLink virtuoso. 2020. https://virtuoso.openlinksw.com/

[28] Eclipse RDF4J. RDF4J. 2020. http://rdf4j.org/

[29] Neumann T, Weikum G. RDF-3X: A RISC-style engine for RDF. Proc. of the VLDB Endowment, 2008,1(1):647−659.

[30] Franz Inc. AllegroGraph. 2020. https://franz.com/agraph/allegrograph/

[31] Ontotext. GraphDB. 2020. http://graphdb.ontotext.com/

[32] Apache TinkerPop. TinkerPop3 documentation v.3.4.8. 2020. https://tinkerpop.apache.org/docs/3.4.8/reference/

[33] Callidus Software Inc. OrientDB-Multi-Model database. 2020. http://orientdb.org/

[34] S1CK. Cypher for apache spark. 2020. https://github.com/opencypher/cypher-for-apache-spark

[35] Gutierrez C, Hurtado CA, Mendelzon AO. Foundations of semantic Web databases. Journal of Computer and System Sciences, 2011,77(3):520−541.

[36] Francis N, Green A, Guagliardo P. Cypher: An evolving query language for property graphs. In: Das G, ed. Proc. of the 2018 Int'l Conf. on Management of Data. New York: ACM, 2018. 1433−1445.

[37] Guo Y, Pan Z, Heflin J. LUBM: A benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agentson the World Wide Web, 2005,3(2-3):158−182.

[38] University of Mannheim. DBpedia. 2020. http://wiki.dbpedia.org/About

附中文参考文献:

[1] 王鑫,邹磊,王朝坤,彭鹏,冯志勇.知识图谱数据管理研究综述.软件学报,2019,30(7):2139−2174. http://www.jos.org.cn/1000-9825/5841.htm [doi: 10.13328/j.cnki.jos.005841]

[13] 邹磊,彭鹏.分布式 RDF 数据管理综述.计算机研究与发展,2017,54(6):1213−1224.

[14] 王童童,荣垂田,卢卫.分布式图处理系统技术综述.软件学报,2018,29(3):569−586. http://www.jos.org.cn/1000-9825/5450.htm [doi: 10.13328/j.cnki.jos.005450]

附 录

1. LUBM数据集上的查询

1) SPARQL 查询

(1) Q1

SELECT ?*X*
WHERE
{?*X* rdf:type ub:GraduateStudent.
?*X* ub:takesCourse
http://www.Department0.University0.edu/GraduateCourse0};

(2) Q2

SELECT ?*X*, ?*Y*, ?*Z*
WHERE
{?*X* rdf:type ub:GraduateStudent.
?*Y* rdf:type ub:University.
?*Z* rdf:type ub:Department.
?*X* ub:memberOf ?*Z*.
?*Z* ub:subOrganizationOf ?*Y*.
?*X* ub:undergraduateDegreeFrom ?*Y*};

(3) Q3

SELECT ?*X*
WHERE
{?*X* rdf:type ub:Publication.
?*X* ub:publicationAuthor
http://www.Department0.University0.edu/AssistantProfessor0};

(4) Q4

SELECT ?*X*, ?*Y*1, ?*Y*2, ?*Y*3
WHERE
{?*X*rdf:typeub:Professor.
?*X* ub:worksFor ⟨http://www.Department0.University0.edu⟩.
?*X* ub:name ?*Y*1.
?*X* ub:emailAddress ?*Y*2.
?*X*ub:telephone ?*Y*3};

(5) Q5

SELECT ?*X*
WHERE
{?*X* rdf:type ub:Person.
?*X* ub:memberOf ⟨http://www.Department0.University0.edu⟩};

(6) Q6

SELECT ?X **WHERE** {?*X* rdf:type ub:Student};

(7) Q11

SELECT ?*X*

WHERE

{?*X* rdf:type ub:ResearchGroup.

?*X* ub:subOrganizationOf ⟨http://www.University0.edu⟩};

(8) Q14

SELECT ?*X*

WHERE {?*X* rdf:type ub:UndergraduateStudent};

2) Cypher 查询

(1) Q1

MATCH

(*x*:GraduateStudent)

−[takesCourse]→

(*y*:GraduateCourse

{*uri*:‘http://www.Department0.University0.edu/GraduateCourse0’})

RETURN *x*;

(2) Q2

MATCH

(*x*:GraduateStudent)−[undergraduateDegreeFrom]→(*y*:University),

(*z*:Department)−[subOrganizationOf]→(*y*),

(*x*)−[memberOf]→(*z*)

RETURN *x*, *y*, *z*;

(3) Q3

MATCH

(*x*:Publication)

−[publicationAuthor]→

(*y*:AssistantProfessor

{*uri*:‘http://www.Department0.University0.edu/AssistantProfessor0’})

RETURN *x*;

(4) Q14

MATCH

(*x*:undergraduatestudent)

RETURN *x*;

2. DBpedia数据集上的查询

1) SPARQL 查询

(1) Q_dbp1

SELECT ?*a*

WHERE

{?*a* ⟨*uri*⟩ “http://dbpedia.org/resource/Alabama”.

?*a* rdf:type ⟨AdministrativeRegion⟩};

(2) Q_dbp2

SELECT ?*a*

WHERE (?*a* rdf:type ⟨Disease⟩);

(3) Q_dbp3

SELECT ?*a*

WHERE (?*a* rdf:type ⟨AdministrativeRegion⟩);

(4) Q_dbp4

SELECT *

WHERE (?*a* rdf:type ⟨TimePeriod⟩);

2) Cypher 查询

(1) Q_dbp1

MATCH

(*a*:AdministrativeRegion{*uri*:'http://dbpedia.org/resource/Alabama'})

RETURN *a*;

(2) Q_dbp2

MATCH (*a*:Disease) **RETURN** *a*;

(3) Q_dbp3

MATCH (*a*:AdministrativeRegion) **RETURN** *a*;

(4) Q_dbp4

MATCH (*a*:TimePeriod) **RETURN** *a*;

刘宝珠(1997－),女,硕士,CCF 学生会员,主要研究领域为知识图谱数据管理.

李思卓(1997－),女,硕士,CCF 学生会员,主要研究领域为知识图谱数据管理.

王鑫(1981－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为知识图谱数据管理,图数据库,大规模知识处理.

张小旺(1980－),男,博士,副教授,博士生导师,CCF 专业会员,主要研究领域为知识图谱.

柳鹏凯(1998－),男,硕士,CCF 学生会员,主要研究领域为知识图谱数据管理.

杨雅君(1983－),男,博士,副教授,主要研究领域为图数据管理,图挖掘.