计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.210500076

基于属性图模型的领域知识图谱构建方法

梁静茹　鄂海红　宋美娜
北京邮电大学计算机学院(国家示范性软件学院)　北京 100876
(liangjingru@bupt.edu.cn)

摘　要　随着大数据时代的到来,各个行业领域需要处理的数据之间的关系数量呈几何级数增长,亟需一种支持海量复杂数据关系表示能力的数据模型,即领域知识图谱。虽然领域知识图谱展现了巨大的潜力,但不难发现目前仍然缺乏成熟的构建技术和平台。如何快速构建出领域知识图谱是一个重要挑战。在对领域知识图谱进行系统的研究后,提出了一种基于属性图模型的领域知识图谱构建方法。该方法对于存储在多种原始业务数据库中的结构化、半结构化数据,通过约定图数据库的数据对接协议、多种图实体模式和关系模式配置方案等方式,完成对应的高质量完整的图谱模式构建;然后将原始数据库的实例数据经过抽取、转换后加载到属性图数据库 HugeGraph 中,完成领域知识图谱的构建。最终,通过对多个数据集进行实验,并使用 Gremlin 语句对知识图谱数据进行测试,验证了所提方法具有完整性和可靠性。

关键词:图数据库;知识图谱构建;领域知识图谱;属性图模型;HugeGraph
中图法分类号　TP392

Method of Domain Knowledge Graph Construction Based on Property Graph Model

LIANG Jing-ru,E Hai-hong and Song Mei-na
School of Computer Science (National Pilot Software Engineering School),Beijing University of Posts and Telecommunications,Beijing 100876,China

Abstract　With the arrival of the big data era,the relationship that needs to be processed in various industries has increased exponentially,and there is an urgent need for a data model that supports the ability to express massive complex relationship,that is,domain knowledge graph. Although the domain knowledge graph has shown great potential,it is not difficult to find that there is still a lack of mature construction technologies and platforms. It still remains an important challenge to construct domain knowledge graph rapidly. After the systematic study of domain knowledge graph,a method is proposed to construct domain knowledge graph based on property graph model. Concretely,for structured and semi-structured data stored in a variety of databases,the method completes the construction of the high-quality graph model by graph database data communication protocol,multiple configuration methods of entity and relation schema,etc. Then,the data from the original database is extracted,transformed and loaded into the property graph database HugeGraph,completing the construction of domain knowledge graph. Finally,experiments on multiple datasets and test results of Gremlin statement show that the proposed method is complete and reliable.

Keywords　Graph database,Knowledge graph construction,Domain knowledge graph,Property graph model,HugeGraph

1　引言

知识图谱是 Google 公司在 2012 年提出来的新概念,它最早被广泛应用于信息搜索领域[1]。知识图谱以结构化的形式描述现实世界中的概念、实体及其关系,提供有价值的结构化信息,是推动数据价值挖掘和支撑智能信息服务的重要基础技术。随着社会和企业对知识图谱构建[2]的需求日益增加,学术界也针对其不断进行研究,如图 1 所示。目前关注的热点在于将机器学习、深度神经网络与自然语言处理信息抽取等技术相结合,自动获取互联网上的非结构化(或半结构化)信息并进行自动化的知识图谱构建。但是对于很多特定领域的知识图谱来说,如果从非结构化文本中自动抽取实体、关系、属性等信息来构建知识图谱,会造成诸多问题,如构建的过程依赖于预先的数据标注、构建后的准确性往往难以保证商用、仍需人工校验来保证可靠性等。

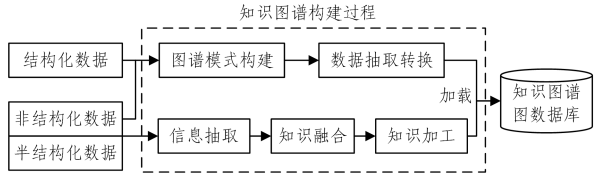

图 1　知识图谱构建技术架构

Fig. 1　Architecture of knowledge graph construction

到稿日期:2021-05-12　返修日期:2021-07-14
基金项目:国家重点研发计划(2018YFB1403501)
This work was supported by the National Key R&D Program of China(2018YFB1403501).
通信作者:鄂海红(ehaihong@bupt.edu.cn)

因此，本文主要关注以行业数据为主的知识图谱（即领域知识图谱）构建。领域知识图谱的原始数据是以结构化和半结构化形式为主，大多存储在各类业务数据库（如 MySQL，MongoDB，Hive 等）中。领域知识图谱的构建主要是将原始数据模型对应的图模式以及原始数据转换而成的节点和边数据加载至图数据库[3]中。其中，图数据库有利于提供完善的查询语句与高效的查询搜索，且更容易表达真实的业务场景。具体来说，属性图模型对于顶点属性和边具备内置的支持，因而被图数据库业界广泛采用，获得了较强的用户认可度；从存储和查询来看，图数据库不仅提供了灵活的设计模式，还提供了高效的关联查询[4]；在图数据库中，可以通过查询实体的边和边上的标签来快速获取与该实体相关联的另一实体，而不用再进行各种表的关联操作，关系查询的效率显著提高[5-10]。

然而，由于传统的数据库（如关系型数据库）和图数据库存在着不小的差异，从原始的各类数据库到图数据库的知识图谱构建仍存在较多的难点。目前，现有的涉及图数据库的数据研究仍较少：Neo4j[11]官方提出了批量导入大量数据的解决方案——neo4j-admin import 工具，需要将数据源整理成 csv 文件，并且需要严格区分节点文件和关系文件，以及规范文件的数据域信息，存在识别文件不正确导致无法导入的问题；Mueller 等提出了一种利用中间表来实现从关系型数据库到 Neo4j 的映射方案[12]，还提供了可视化界面以展示关系结构，但是仍然局限于利用中间表处理关系情况；Unal 等提出了一种基于层级化结构数据的从关系型数据库迁移到图数据库的方法[13]，还比较了关系数据库和图形数据库的数据访问过程和性能；Virgilio 等在 2013 年提出了利用模式和约束实现从关系型数据库到图数据库的转换方法[14]，在 2014 年提出了一个将数据从关系型数据库 r 自动迁移到图数据库管理系统 g 的工具[15]，并将 r 上的查询转换为 g 上的图查询；Anzum等提出了数据表到图的交互式映射系统[16]，使得用户能够通过拖拽等方式在屏幕上绘制图；Serin 等[17]提出一种针对公共交通网络的图构建方法，将数据从关系型数据库转移到图数据库中进行管理，使得城市交通服务分析更加高效。

但是上述研究仍存在不少缺陷：均仅局限于将关系型数据库中的结构化数据构建成图谱，或者是使用单一的方式建立图，缺乏多样的图模式配置方案，抑或是仅针对特定的数据集构建图谱和网络。因此，本文提出了一种基于属性图模型的领域知识图谱构建方法，保证构建高质量的、完整的领域知识图谱。本文的主要贡献在于：

（1）首次提出了图数据库数据对接协议。由于传统的数据库和图数据库天然存在的异构性，数据类型与格式都存在不小的差异，此外，各类原始数据库中还存在数据约束，来制约和限定数据库的数据模型以及状态的变化。这些都会影响领域知识图谱构建的完整性和一致性。因此，本文首次提出了一种图数据库的数据对接协议（Graph Database Data Communication Protocol，GDBDCP），保证构建前后数据不出错，形成高质量的领域知识图谱。

（2）首次提供了多种图实体模式和关系模式配置方案。在任何数据库中，数据模型对于确定查询的逻辑和存储中的数据结构非常重要。但是存储在各类原始数据库中的数据模式复杂多样，数据之间联系的表示方式各不相同，且与图数据库的模式（包含实体和关系模式）存在天壤之别，这会影响领域知识图谱构建的正确性、有效性和相容性。因此，本文首次提出了多种图实体模式和关系模式配置方案（Multiple Configuration Methods of Entity and Relation Schema，MCMERS），以满足不同数据模式的知识图谱构建需求，保证领域知识图谱建模的准确、有效。

（3）实现了将多种原始业务数据库（多种形式）的数据构建成领域知识图谱。本文通过对多种数据源的数据模式与存储数据进行分析解析以及参照本文提出的多种模式配置方案，实现了将存储在典型业务数据库 MySQL，MongoDB 和 Hive 中的结构化、半结构化数据经过抽取、转换后加载到属性图数据库中，最终完成构建对应的领域知识图谱。

2 整体框架

如图 2 所示，本文提出的基于属性图模型的领域知识图谱构建方案主要由以下 3 个部分组成：1）建立外部原始数据库到图数据库的数据对接协议；2）通过多种图实体模式和关系模式配置方案构建以属性图模型为表现形式的领域知识图谱；3）将原始数据库的实例数据经过抽取、转换后加载到属性图数据库，最终完成领域知识图谱的构建。

首先建立外部原始数据库到图数据库的数据对接协议，解决数据类型与格式异构等问题，保证领域知识图谱构建前后的一致性与完整性。

然后进行领域知识图谱的模式（schema）构建。1）构建所有的实体模式。对原始的业务数据库表按照对接协议建立所有的属性字段（在 HugeGraph 中为 PropertyKey），表的所有属性会映射为其实体类型（在 HugeGraph 中为 VertexLabel）的所有属性。以图 2 中的 Person 表为例，Person 表的所有属性会映射为 Person 实体类型的所有属性。2）关系模式的构建。基于中间表外键或基于字段配置的方案建立关系模式，提取出关系中的指向信息，明确关系的头尾节点信息。以图 2 中关系 Publish 为例，首先建立一个 Publish 的关系类型，然后根据基于字段配置的方法（如图 2 中 Paper 表的 Author_id 字段指向 Person 表的 ID 字段），或基于中间表外键的方法（如图 2 中 Publish 表中 Paper_id 字段指向 Paper 表的 ID 字段，Person_id 字段指向 Person 表的 ID 字段），确定出 Publish 关系指向信息，从 Paper 实体指向 Person 实体。3）数据的抽取、转换。生成所有符合图数据库规范及图模式的节点（vertex）和边（edge），加载至属性图数据库中。

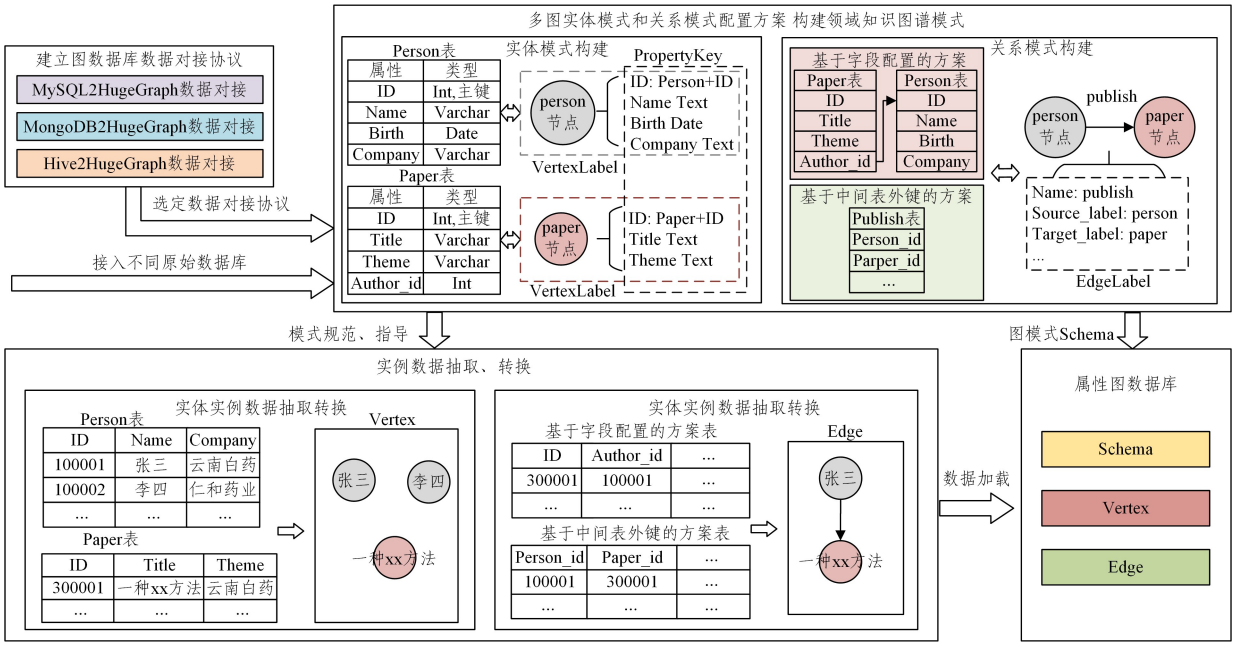

图 2 基于属性图模型的领域知识图谱构建架构图

Fig. 2 Architecture of domain knowledge graph construction based on property graph model

3 图数据库数据对接协议(GDBDCP)

本节将介绍如何规定外部原始数据库到图数据库的数据对接协议。图数据库选取开源数据库 HugeGraph,其设计理念借鉴了 Janusgraph 和 Titan[18],具备设计规范的数据模型,并使用 Gremlin 进行检索和计算,可以很好地为图谱数据提供检索和查询计算。

在外部原始数据库中,领域数据通常会以表/文档等形式存储,因此本文设计对接协议:原始数据库表字段名称在图数据库中对接为属性名称,表字段类型对接为该种属性的类型,每一行用来描述该属性的具体值,对接为对应实体的该属性的具体值。为了保证领域知识图谱数据构建前后的一致性和完整性,仍需要先解决数据类型与格式异构等问题。本文通过对 MySQL,MongoDB 和 Hive 中存在的属性类型进行统计分析后,在最大程度保留元数据的基础上,定义了表 1—表 3 的图数据库数据对接协议。

表 1 MySQL-HugeGraph 数据对接

Table 1 MySQL-HugeGraph data communication

type in MySQL	type in Hugegraph
char,varchar,text,enum,year,set	Text
int,smallint,tinyint	Int
bigint	Long
float,double,decimal	Double
datetime,date,timestamp	Date
boolean	Boolean

表 2 MongoDB-HugeGraph 数据对接

Table 2 MongoDB-HugeGraph data communication

type in MongoDB	type in Hugegraph
string,symbol,ObjectID	Text
int	Int
double	Double
array	Text
date,timestamp	Date
boolean	Boolean

表 3 Hive-HugeGraph 数据对接

Table 3 Hive-HugeGraph data communication

type in Hive	type in Hugegraph
string	Text
int,smallint,tinyint	Int
bigint	Long
float,double	Double
array	Text
timestamp	Date
boolean	Boolean

特别地,本文对 MongoDB 和 Hive 中的 Array 数据类型进行了特殊处理,Array 用于将数组、列表或多个值存储为一个键。在对接到图数据库之后,对于作为属性的 Array 来说,我们将 Array 数据用逗号分隔开存为字符串,表示其属性信息。

除此之外,原始数据的一些字段约束如 PRIMARY KEY,NOT NULL,UNIQUE 以及索引列等信息需要保留,因此本文还增加了字段约束的对接协议。

PRIMARY KEY:该约束规定实体表中该字段为主键。相应地,在图数据库中建立一个实体类型时,将该字段作为该实体的 id,如指定图数据库 id 策略为 useCustomizeStringId 策略,后续利用 PRIMARY KEY 进行 id 的赋值。

NOT NULL:该约束规定实体表中该字段不能为空。相应地,在图数据库中建立一个实体类型并配置其属性时,可以规定某些字段为非空字段,使用 notNull 字段限定,如式(1)创建一个实体类型 v 并使属性 prop2 不可为空。

schema.vertexLabel("v").properties('prop1',

'prop2',).notNull('prop2').ifNotExist().create() (1)

UNIQUE:该约束规定实体表中该字段具有唯一性。相应地,在图数据库中使用 Unique 来支持属性值唯一性约束,限定属性的值不重复,如式(2)通过索引信息为实体类型 v 的属性 id 增加 Unique 限制。

schema.indexLabel("vById").onV("v").by("id").
unique().ifNotExist().create() (2)

索引列 COLUMN_NAME：表示该字段(列)为索引列，可快速访问数据库表中的特定信息。相应地，在图数据库中配置实体类型的属性时，可以使用 IndexLabel 来定义索引类型，描述索引的约束信息，如式(3)为实体类型 v 的属性创建索引，通过 Secondary 实现 name 属性的精确匹配索引。

schema.indexLabel("vByName").onV("v").
by("name").secondary().ifNotExist().create() (3)

通过上述图数据库对接协议的建立，可以保证在图数据构建的前后，最大程度地保留实体属性的类型和数据约束信息，数据在构建图谱前后不出错，从而保证领域知识图谱构建前后的一致性与完整性。

4 多图实体模式和关系模式配置方案(MCMERS)

本节将介绍如何通过多种图实体模式和关系模式配置方案来规定图数据库的图模式。HugeGraph 以属性图模型为表现形式，实体类型 VertexLabel 可以包含零个或多个属性 PropertyKey，关系类型 EdgeLabel 则指定了两个 VertexLabel 分别为起始点和终点。本节会分别对 VertexLabel 和 EdgeLabel 进行构建，即模式构建。

本文在第 3 节建立了图数据库数据对接协议，因此将原始数据库表(实体表)的表字段按照对接协议对接成图数据库的实体类型及其拥有的属性，从而保证实体模式的完整一致。对于关系模式，本文提出了基于中间表外键与基于字段配置的关系模式定义方法，保证关系模式的完整一致性，分别解决原始数据库中以两种方式存储关系数据的场景：1)通过中间表(关系表)存储关系；2)通过指定实体表 A 中的特定字段关联实体表 B 的字段。本文通过对多个数据库的存储模式和数据进行分析发现，其对关系的设计也有所不同：对于 MySQL 来说，其数据模式约束较多，外键清晰，关系明确，存储的是定义明确的实体表与中间关系表；Hive 的数据模式与 MySQL 类似，但是没有数据模式的约束如 MySQL 外键等，所以关系与属性都包含在一张实体表中；MongoDB 作为一个基于分布式文件存储的数据库，支持无模式的数据建模方式，大多存储的是实体类型较少、关系较少、实体属性较多以及带有时序性的数据。因此，本文设计多种关系模式构建方法以满足不同原始数据库构建知识图谱的需要。

4.1 实体模式构建

本文对 MySQL，MongoDB 和 Hive 这 3 种数据源的数据模式进行分析发现，MySQL 与 Hive 数据源较类似，均可通过 describe 操作得到表字段；MongoDB 由于支持无模式的数据建模方式，需要寻求另一种方式进行实体模式构建。下面将详细介绍不同数据源、不同数据格式的实体模式构建方法。

4.1.1 MySQL Hive 结构化信息实体模式的构建

对于 MySQL 和 Hive，我们可以获得选定的实体表的元数据，对照图数据库数据对接协议构建实体模式(实体标签和属性等)，如图 3 所示。

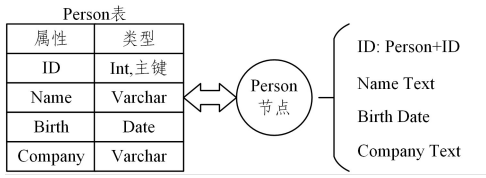

图 3 实体模式构建框架

Fig. 3 Framework of entity schema construction

对应图数据库的流程为创建 PropertyKey(规范顶点和边的属性的约束)和创建 VertexLabel 定义顶点类型，如式(4)—式(6)创建了 name，birth，company 3 个属性类型，分别为 Text 类型、Date 类型和 Text 类型；式(7)创建了拥有上述 3 类属性的实体类型 person。

schema.propertyKey("name").asText().ifNotExist().
create() (4)

schema.propertyKey("birth").asDate().ifNotExist().
create() (5)

schema.propertyKey("company").asText().ifNotExist
().create() (6)

schema.vertexLabel("person").useCustomizeStringId
().properties("name""birth""company").ifNotExist().
create() (7)

4.1.2 MongoDB 半结构化信息实体模式构建

MongoDB 支持无模式的数据建模方式，即可以按需进行模式的添加或修改。因此在探索 MongoDB 指定集合的数据模式时，我们需要遍历该集合下的数据记录行的字段，将所有字段组成的并集作为该集合的属性，如图 4 所示。

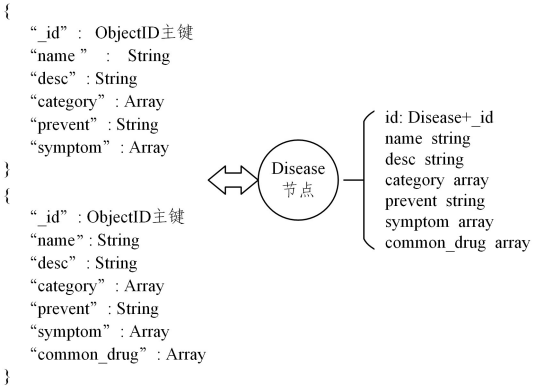

图 4 MongoDB 实体模式的构建框架

Fig. 4 Framework of entity schema construction of MongoDB

例如，对于 disease 集合来说，存在属性字段为 id，name，dese，category，prevent，symptom 的数据行和属性字段为 id，name，desc，category，prevent，symptom，common_drug 的数据行，将其做并集，即最后的 disease 实体拥有 id，name，dese，category，prevent，symptom，common_drug 属性。

4.2 关系模式构建

本文对 MySQL，MongoDB 和 Hive 这 3 种数据源的数据模式进行分析发现，MySQL 主要通过增加中间表(关系表)以及外键约束来存储关系数据；而 MongoDB 和 Hive 主要通过指定实体表 A 中的特定字段关联实体表 B 的特定字段来存储关系数据。为了保证形成高质量的关系模式，本文提出

了两种关系模式构建方法：基于中间表外键与基于字段配置的关系模式定义方法。

4.2.1 基于中间表外键的关系模式构建

在基于中间表外键的模式下，原始业务数据库的数据表会严格分为实体表和关系表。以中间表(关系表)的方式去建立两表(两种实体)之间的关系，通常会使用复合主键的形式，即两个字段分别对应其他两张实体表的主键。因此该设计模式适用于 MySQL 的关系模式构建。图数据库的关系模式建立会从关系表入手，直接将该表映射为图数据库的一种关系类型，提取该表通过外键关联的两个实体表，将其映射为关系的头(source)实体类型和尾(target)实体类型，其他字段映射为关系的属性。具体的流程如图 5 所示。

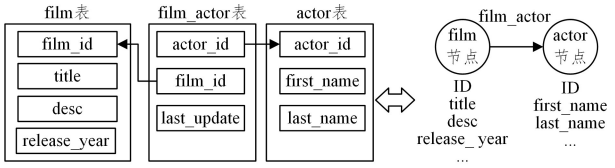

图 5 基于中间表外键的关系模式构建

Fig. 5 Relation schema construction based on intermediate table and foreign key

film 表包含{ film_id, title, description, release_year }的属性类型，且 film_id 为其主键；actor 表包含{ actor_id, first_name, last_name }的属性，id 同样为其主键；film_actor 表包含{ actor_id, film_id, last_update }两个属性字段，并且 actor_id 和 film_id 作为外键分别关联 actor 表的 actor_id 和 film 表的 film_id。在构建模式过程中，首先将 T1, T2 实体表构建为图数据库的实体模式；对于 T3，提取出表中的外键约束关联的实体表信息以及表中其他的属性字段，如式(8)创建关系类型：

```
schema.edgeLabel("film_actor").sourceLabel("film").
targetLabel("actor").properties("last_update").ifNot-
Exist().create()                                    (8)
```

4.2.2 基于字段配置的关系模式构建

基于字段配置的关系模式构建，可以让构建过程的操作更加细粒度，图谱的结构和数据也更加符合用户的直观预期，但同时也需要一些人工干预，通过指定实体表中的某些特定字段来关联其他实体表。因此该设计模式适用于 MongoDB 和 Hive 的关系模式构建。图数据库的关系模式建立会从用户选择的特定字段入手，通过指定其关联的实体类型为其设置一个额外的关系。具体的流程如图 6 所示。

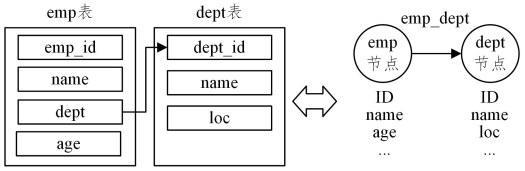

图 6 基于字段配置的关系模式构建

Fig. 6 Relation schema construction based on configuration

图 6 将 emp 表的 dept 属性关联到 dept 表的 dept_id 属性，建立两个实体之间存在的关系 emp_dept。

同时，本文研究发现，MongoDB 存储的大多是实体类型较少、实体属性较多以及带有时序性的数据，且关系经常以 ArrayList 的形式蕴藏在属性字段中。因此在采用基于字段配置的方案时，对于用户选择的特定字段，我们不会指定其关联的实体表(实体类型)，而是对应该字段创建一个辅助类的实体类别(该实体类别并无属性，只存在 name 属性)，从而建立主实体类型与辅助类实体类型的关系，具体如图 7 所示。

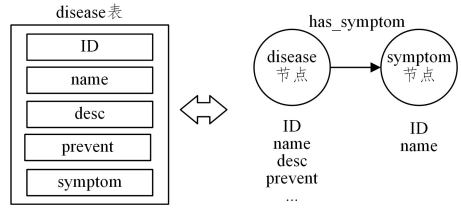

图 7 MongoDB 基于字段配置的关系模式构建

Fig. 7 Relation schema construction based on configuration of MongoDB

主实体标签为疾病 disease，通过指定属性字段 symptom 创建其关联的辅助类实体标签 symptom，以及 disease 与 symptom 存在的关联关系 has_symptom。

5 实例数据抽取、转换和加载

本节将介绍在属性图数据库的图模式构建后，如何将原始数据库的实例数据抽取、转换成符合属性图数据库规范的节点和边，最后加载到属性图数据库 HugeGraph 中，完成领域知识图谱的构建。由于转换后的节点和边加载到 HugeGraph 这一过程较为简单，本节主要介绍实例数据的抽取过程和转换过程。

5.1 实例数据抽取

在本文中，领域实例数据的抽取指从各个不同的原始数据库将数据抽取出来，等待后续的转换与加载。在抽取的过程中需要根据第 4 节的模式配置方案来挑选不同的抽取方法，以确保转换后的实例数据满足图模式，具体分为基于中间表外键的数据抽取方式和基于字段配置的数据抽取方式。

在基于中间表外键的数据抽取下，第一步是抽取实体实例数据，遍历所有的实体表，抽取表中的所有实体数据(行数据)；第二步是抽取关系实例数据，遍历中间表外键指向的实体表，根据其中的外键信息建立外键列-主键列的缓存，如缓存〈film_actor 中 film_xx 指向的 film 列，film. id〉和〈film_actor 中 actor_xx 指向的 actor 列，actor. id〉，然后遍历中间表数据“film_信息---actor_信息”查找缓存，从而抽取出〈film. id---actor. id〉格式的关系实例数据。

在基于字段配置的数据抽取下，以 emp 表和 dept 表为例，emp 表的 dept 字段指向 dept 表的 name 字段，则预先建立一个从 emp 节点指向 dept 节点的 emp_dept 的关系类型。如果在建立实体的同时建立边，会出现尚无对应节点的情况，因此有必要先建立所有的实体，再根据关系的信息来缓存节点数据，建立关系。具体的抽取关系实例数据的步骤如下。

(1)在遍历 emp 和 dept 表时，对 dept 表缓存〈dept_name, dept_id〉，对 emp 表缓存〈emp_id, dept〉关系信息。

(2)抽取关系数据，执行下述操作：

1)遍历所有的缓存信息，如 dept 的 name 属性值及其 id 的对应关系、emp 的 dept 属性值及其 id 的对应关系；

2)遍历缓存的映射关系，抽取出真正的关系，即数据格式

为〈emp. emp_id---dept. dept_id〉的关系实例数据。

5.2 实例数据转换

本文中，领域实例数据的转换指将抽取好的实体数据和关系数据，转换成符合图数据库 HugeGraph 规范的节点和边。

由于在图数据库中使用的是全局命名空间且不允许同时存在两个索引值相同的节点，为了保证数据在构建前后的唯一性，本文针对抽取后的实体数据，提供模式“tableName-primaryValue”来解决节点的唯一性，进而保证后期的节点查询可靠性。具体地，对于实体节点的标识使用“useCustomId”即自定义 Id 的策略，通过主键和表名的复合形式设置实体 Id，其他字段值则对应实体的属性值。对于抽取后的关系数据，将头实体 Id 和尾实体 Id 分别设置为 outV 和 inV，其他字段值则对应边的属性值。

6 实验分析

本文实验使用的设备信息为：处理器 2.4 GHz Intel Core i5，内存 8 GB 2133 MHz LPDDR3，系统为 MacOS。使用 Java 作为开发语言实现本文提出的方法。

本文通过对不同原始数据库、不同量级的相关领域数据集进行实验，进而验证本文提出的基于属性图模型的领域知识图谱构建方案的可行性，同时针对不同业务场景执行 Gremlin 查询语句，进而验证数据集的可靠性。在本实验中，对于原始数据源 MySQL 使用了 MySQL 官方提供的 sakila 数据集，对于 MongoDB 使用了医药数据集[19]，对于 Hive 使用了 YouTube 视频统计与社交网络数据集[20]，分别进行验证。

6.1 MySQL 构建方案验证

本实验验证基于 MySQL 官方提供的 sakila 数据集，这是 MySQL 官方提供的一个模拟 DVD 租赁信息管理的数据库，包含了 actor，film，staff，store 等多个用于描述 DVD、员工以及仓库的相关信息的实体表，以及 film_actor，film_category，rental 等多个用于描述电影与演员的关系、租借的记录等的中间表。

通过对不同数量级的数据进行知识图谱构建测试，对耗时进行统计，结果如表 4、表 5 所列。

表 4 MySQL 实体构建耗时统计

Table 4 MySQL entity construction time statistics

数据表	实体数量	模式构建速度/s	数据转换速度/s
category	16	0.073	0.089
actor	200	0.092	0.249
film	1 000	0.126	0.629

表 5 MySQL 关系构建耗时统计

Table 5 MySQL relation construction time statistics

数据表	关系数量	模式构建速度/s	数据转换加载速度(无缓存)/s	数据转换加载速度(缓存)/s
film_catology	1 000	0.066	20.536	0.603
film_actor	5 462	0.068	25.532	3.342

可以看出，本文提出的方法建立图数据库的 Schema，每次耗时基本不足 0.1 s。主要的耗时在于实体/关系实例数据转换加载到图数据库 HugeGraph：在单个线程的情况下，1 000个节点加载进图数据库仅需 0.6 s；默认情况下读取关系表边数据的插入速度较慢，转换 5 462 条边需要约 25 s；在提前缓存了头尾实体信息的情况下(如 5.1 节所述，利用 Redis 进行数据缓存)，数据耗时大幅减少，转换 5 462 条边需要约 3 s。

知识图谱构建完成后，同样需要对该知识图谱构建前后的数据一致性和完整性进行验证，本文提供数据检验算法，将 MySQL 的实体表的每行数据(构建前)与 HugeGraph 中该实体类型的每个节点(构建后)进行对比，将 MySQL 的关系表的每行数据与 HugeGraph 的该关系类型的每条边数据进行对比，得到数据一致性分数。具体算法如算法 1 所示。

算法 1 知识图谱构建前后的数据一致性检验算法

输入：原始数据库实体表行数据列表 OriginData，HugeGraph 节点列表 GraphData

输出：实体数据一致性指标分数 score1

初始化：num1←0，num2←0

```
1. for i←0 to length[OriginData] do
2.   /* 检查原始数据是否在图数据列表中 */
3.   if OriginData[i]. id exist in GraphData. id then
4.     /* 寻找指定 id 的图节点 */
5.     node←findID(OriginData[i]. id)
6.     flag←true
7.     /* 检查第 i 条数据的各项属性 j 与图数据列表中对应节点的属性是否对应 */
8.     for j←0 to length[OriginData[i]] do
9.       if check(OriginData[i][j],node) is false then
10.      flag←false
11.    end if
12.  end for
13.  if flag is true then
14.      num1←num1+1
15.  end if
16.  end if
17. end for
18. score←num1/length[OriginData]
```

本文先检验原始数据库实体表的每行数据是否有节点与之对应；再检验 HugeGraph 中的每个节点是否有实体表行数据与之对应；然后检验原始数据库关系表的每行数据是否有边一一对应。本实验输出数据一致性指标分数均为 100%，证明了知识图谱构建前后的数据具备完整性和一致性。

根据一些常见的业务场景，我们对该数据集使用下述语句进行了实验验证，并记录了其返回值，结果如表 6 所列。

表 6 MySQL 结果验证

Table 6 MySQL result verification

业务场景	Gremlin 语句	返回值
查询所有 film 个数	g. V( ). hasLabel(“film”). count()	[1 000]
随机查询 3 部电影	g. V( ). hasLabel(“film”). limit(3)	[film-400, film-662, film-884]
参演 ORDER BETRYED 电影的所有演员	g. V(). hasLabel(“film”). has(“title” “ORDER BETRYED”). both(). path()	[actor-76, actor-100, actor-113, actor-120, actor-157, actor-184]
ORDER BETRYED 电影的评级	g. V(). hasLabel(“film”). has(“title” “ORDER BETRYED”). values(“rating”)	[“PG-13”]

6.2 MongoDB 构建方案验证

本实验验证一个开源的医药数据集，包含一张主实体表 disease 用于描述疾病、诊断检查项目、医疗科目、药品、食物、疾病症状的相关信息，以及这张主实体表蕴含的疾病所属科目、症状、并发疾病、所需检查、常用药品、推荐药品、宜吃食物、忌吃食物、推荐食谱的关系。

通过对该数据集进行测试(实体与关系构建一同计算)，对耗时进行统计，结果如表 7 所列。

表 7 MongoDB 知识图谱构建的耗时统计

Table 7 MongoDB KG construction time statistics

数据集	实体总数量	关系总数量	模式构建速度/s	全量数据转换加载速度/s
疾病数据集	26910	281735	0.403	40.450

由表 7 可以看出，本文对该数据集进行图数据库模式构建(包括上述的 6 种实体类型和 9 种关系类型)仅耗时 0.4 s，全量实体与关系数据(共计 26910 节点，281735 条边)转换加载到图数据库中大约需要 40 s。

本文检验了 MongoDB 数据库主实体表的每行数据是否与 HugeGraph 节点和边一一对应(过程类似算法 1，这里不再赘述)。本实验输出数据一致性指标分数均为 100%，证明了知识图谱构建前后的数据具备完整性和一致性。

同样，我们使用下述语句进行实验验证并记录其返回值，结果如表 8 所列。

表 8 MongoDB 结果验证

Table 8 MongoDB result verification

业务场景	Gremlin 语句	返回值
查询所有 disease 个数	g. V ( ). hasLabel ("disease"). count()	[8807]
查询关系为疾病推荐食谱的个数	g. V ( ). hasLabel ("recommand_eat"). count()	[40221]
疾病-百日咳相关的所有症状、药品、食物等信息	g. V ( ). hasLabel ("disease"). has (id, "disease-百日咳"). both (). path()	如图 8 所示
疾病-皮肤过敏的收费标准	g. V ( ). hasLabel ("disease"). has (id, "disease-皮肤过敏"). values ("cost_money")	["不同医院收费标准不一致，市三甲医院约为(1000—5000 元)"]

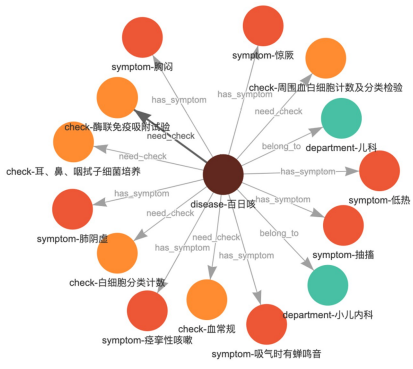

图 8 百日咳相关症状、药品等查询结果(HugeGraph)

Fig. 8 Query results of pertussis related symptoms, drugs, and other information in HugeGraph

6.3 Hive 构建方案验证

本实验验证一个开源的 Youtube 视频统计与社交网络数据集，它是西蒙弗雷泽大学在 2008 年爬取的数据，包含 video 表和 uploader 两张实体表，其中 video 表中的"上传"字段为 uploader 的名称，表示 video 与 uploader 建立了"上传"的关联关系。

对该数据集进行领域知识图谱构建测试，对耗时进行统计，结果如表 9 和表 10 所列。

表 9 Hive 实体构建耗时统计

Table 9 Hive entity construction time statistics

数据表	实体数量	模式构建速度/s	数据转换加载速度/s
video	233607	0.064	28.774
uploader	1062322	0.042	99.876

表 10 Hive 关系构建耗时统计

Table 10 Hive relation construction time statistics

数据表	关系数量	模式构建速度/s	数据转换加载速度/s
video_ uploader	233607	0.034	19.375

本文检验了 Hive 数据库表的每行数据与 HugeGraph 节点和边是否一一对应(过程类似算法 1，这里不再赘述)。本实验输出数据一致性指标分数均为 100%，证明了知识图谱构建前后的数据具备完整性和一致性。

通过对不同原始数据库的不同数据集进行测试，并与最新的工作进行对比可发现，本文方法应用场景更广，且实验在测试机器上的效率甚至可以达到 10000 节点/秒(多线程情况下)，并且构建的领域知识图谱具有较强的可靠性，具有速度快、可靠性高的优势。

结束语 本文通过对领域知识图谱进行系统研究，设计了一种基于属性图模型的领域知识图谱构建方法，对于存储在多种业务数据库中的结构化、半结构化数据，通过约定图数据库的数据对接协议、多种图实体模式和关系模式配置方案等方式，完成对应的完整的高质量图谱模式构建；然后将原始数据库的实例数据经过抽取、转换后加载到属性图数据库，完成领域知识图谱的构建。目前学术界对于此类方法的研究较少，大多局限在特定的数据集或特定的图数据库如 Neo4j，而且缺乏多样的图模式配置方案，难以保证数据的完整性和一致性。而本文提出的方法具有通用性，提出的多种图模式配置方案和图数据库数据对接协议保证了知识图谱构建数据的完整一致性，适用于存储在不同数据库、不同格式的原始领域数据，且适用于基于 TinkerPop 3 框架的多类图数据库后端，具有一定的独创性。本方法仍需进一步改进，如支持更多图谱模式的配置以及扩大数据源的适配面。

参 考 文 献

[1] SINGHAL A. Introducing the knowledge graph: things, not strings[J]. Official Google Blog, 2012, 5: 16.

[2] HANG T T, FENG J, LU J M. Knowledge Graph Construction Techniques: Taxonomy, Survey and Future Directions[J]. Computer Science, 2021, 48(2): 175-189.

[3] ROBINSON I, WEBBER J, EIFREM E. Graph Databases[M]. Cambridge: O'Reilly Media, Inc, 2015: 12-20.

[4] LI J Y. Graph database white paper and basic functional stan-

dards[R]. Beijing:China Academy of Information and Communications Technology,2019.

[5] NEO4J STAFF. The Database Model Showdown:An RDBMS vs. Graph Comparison[EB/OL]. (2015-08-03)[2021-05-01]. https://neo4j. com/blog/database-model-comparison.

[6] OZGUR C,COTO J,BOOTH D. A comparative study of network modeling using a relational database (eg Oracle,mySQL,SQL server) vs. Neo4j[J]. International Journal of Engineering Research,2018,8(7):27-32.

[7] MAGORZATA P W,RYKOWSKI D. Comparison of Relational,Document and Graph Databases in the Context of the Web Application Development[M]. Swiss: Springer International Publishing,2016:3-13.

[8] FOSIC I,SOLIC K. Graph database approach for data storing, presentation and manipulation[C]// 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics. Opatija: IEEE Press, 2019: 1548-1552.

[9] SHOLICHAH R J,IMRONA M,ALAMSYAH A. Performance Analysis of Neo4j and MySQL Databases using Public Policies Decision Making Data[C]// 2020 7th International Conference on Information Technology,Computer,and Electrical Engineering (ICITACEE). Semarang:IEEE Press,2020:152-157.

[10] BATRA S,CHARU T. Comparative analysis of relational and graph databases[J]. International Journal of Soft Computing and Engineering (IJSCE),2012,2(2):509-512.

[11] The Neo4j Team. The Neo4j Manual v3. 4[EB/OL]. (2018-05-16)[2021-05-01]. https://neo4j. com/docs/developer-manual/current.

[12] MUELLER W,IDZIASZEK P. Mapping and visualization of complex relational structures in the graph form using the Neo4j graph database[C]// Proceedings of Eleventh International Conference on Digital Image Processing. Guangzhou,2019:456-462.

[13] UNAL Y,OGUZTUZUN H. Migration of data from relational database to graph database[C]// Proceedings of the 8th International Conference on Information Systems and Technologies. New York:Association for Computing Machinery,2018:1-5.

[14] VIRGILIO R D,MACCIONI A,TORLONE R. Converting relational to graph databases[C]// First International Workshop on Graph Data Management Experiences and Systems (GRADES'13). New York: Association for Computing Machinery, 2013: 1-6.

[15] VIRGILIO R D,MACCIONI A,TORLONE R. R2G:a Tool for Migrating Relations to Graphs[C]// Proceedings of International Conference on Extending Database Technology. Athens: OpenProceedings,2014:640-643.

[16] ANZUM N. Systems for Graph Extraction from Tabular Data[D]. Waterloo:University of Waterloo,2020.

[17] SERIN F,METE S,GUL M,et al. Mapping Between Relational Database Management Systems and Graph Database For Public Transportation Network[C]// 21st International Research/Expert Conference "Trends in the Development of Machinery and Associated Technology". Karlovy Vary,2018:209-212.

[18] THUTMOSE.「JanusGraph 与 HugeGraph」图形数据库-技术选型-功能对比[EB/OL]. (2019-03-25)[2021-05-01]. https://blog. csdn. net/lovebyz/article/details/88800363.

[19] LIU H Y. QABasedOnMedicaKnowledgeGraph[EB/OL]. (2018-10-04)[2021-05-01]. https://github. com/liuhuanyong/QASystemOnMedicalKG.

[20] XU C,DALE C,LIU J. Statistics and Social Network of YouTube Videos[C]// 16th International Workshop on Quality of Service. Enskede:IEEE Press. 2008:229-238.

LIANG Jing-ru, born in 1997, postgraduate, is a student member of China Computer Federation. Her main research interests include knowledge graph and graph database.

E Hai-hong, born in 1982, Ph. D, associate professor, is a member of China Computer Federation. Her main research interests include big data platform, cloud computing and microservice architecture.

(责任编辑:李亚辉)