计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.201100073

关系型数据库向图数据库的转换方法

鄂海红　韩鹏昊　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

摘　要　由于关系型数据库和图数据库存储模式的天然差别,将关系型数据库中的数据转存到图数据库的过程中,需解决对于关系的定义、节点唯一性以及保留原数据库约束信息的主要问题。针对上述问题,提出了一种关系型数据库向图数据库转换的方法。首先通过自定义或使用已有主键,并结合数据库表名的唯一性,解决了节点唯一性的问题;通过不同的配置方案,最大化保留了原关系型数据库的约束信息;然后提出了基于配置与中间表的边定义方法(Edge Definition Method based on Configuration and Intermediate Table,EDCIT),针对多种类型的数据库提供不同关系的映射方案,解决了转换过程中对于关系的定义。最终,通过对多个数据集进行实验,并使用 Gremlin 语句对转换后的数据进行测试,验证了转换后的数据具有完整性和可靠性。

关键词:图数据库;关系型数据库;跨库数据交换;Hugegraph;Gremlin
中图法分类号　TP392

Conversion Method from Relational Database to Graph Database

E Hai-hong, HAN Peng-hao and SONG Mei-na
School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing 100876, China

Abstract　Due to the differences between the storage mode of relational database and graph database, during the process of transforming data in relational database to graph database, it is necessary to solve the main problems of edge definition, vertex uniqueness and retention of original database constraint information. To solve the above problems, a method of transforming relational database to graph database is proposed. Firstly, by customizing the existing primary key, combined with the uniqueness of the table name, the problem of ensuring the uniqueness of the vertex is solved; through different configuration schemes, the constraint information of the original relational database can be maximized. Then, the edge definition method based on configuration and intermediate table (EDCIT) method is proposed, it provides different edge mapping solutions for multiple types of databases and solves the definition of edges during the transformation. Finally, through experiments on multiple data sets, and using Gremlin statement to test the transformed data, it verifies the integrity and reliability of the transformed data.

Keywords　Graph database, Relational database, Cross-database data exchange, Hugegraph, Gremlin

1　引言

大数据时代,高关联度数据的数据量不断增加,这些数据的适用场景不断增多,对于关联数据的存储和分析的需求也不断增加,而这正是图数据库[1]及图挖掘算法的用武之地,它有助于我们理解海量数据,针对关系进行更复杂的分析,从而为洞察和创新提供了巨大的潜力[2],因此将海量数据存入图数据库中进行存储并对关联数据进行深度挖掘是大势所趋。以往开发者会使用关系型数据库如 MySQL 对领域数据进行存储,这类关系型数据库诞生较早,但并不适合存储关联度较大的数据如社交关系网络[3-4]。在关系型数据库中,通常会通过中间表或外键的方法存储实体之间的关系,但随着数据量的增多,某些场景下的查询如"查询某社交网络中一个人的朋友的朋友…"会产生大量的联表查询操作,这也会使 MySQL 在查询过程中降低较多的性能。另一方面,图数据库通过属性、节点和关系的方式,使得关系、实体的查询变得更为容易[5-10]。但由于关系型数据库和图数据库天然存在的异构性,将已有的关系型数据库中的数据存储至图数据库中仍存在较多的难点。据笔者的了解,已有的两者之间转换的研究仍较少,且大都是针对某个数据集提出的一套方案,抑或是针对于特定的图数据库即 Neo4j,这类方案都具有一定的局限性。

根据现有涉及图数据库的数据导入和数据转换的研究来看,Neo4j 官方提出了批量导入节点和关系的解决方案 Load CSV 和 neo4j-admin;Mueller 等从外键角度提出了一个简易的转换方法[11];Unal 等研究了层级化数据的转换方法,并比

到稿日期:2020-11-09　返修日期:2021-01-04　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划课题(2018YFB1403501)
This work was supported by the National Key Research and Development Program of China(2018YFB1403501).
通信作者:鄂海红(ehaihong@bupt.edu.cn)

较了使用两类数据库进行查询时存在的性能差别[12]。诸如此类仍有一些研究[13-16]，但大都局限于特定的数据集，使用固定的方式建立关系图（如仅使用外键），同时也都局限于指定的图数据存储后端。

总的来看，要实现将数据从关系型数据库向图数据库的转换主要需要解决以下几个难点：数据库表和节点关系的映射、灵活解决节点唯一性的问题、映射数据库中的数据类型并转存其余约束信息等。因此，本文的贡献在于使用下述方法解决了这些问题。

（1）实现数据库表数据的映射

通过人工额外指定的方式，提供基于配置和中间表两种形式的边定义模式，进而实现数据库表的元数据和图数据库 Schema 的映射，最终将表中数据通过指定好的 Schema 映射为具体的节点和关系。

（2）保证节点映射过程中的唯一性

在图数据库中不允许同时存在两个索引值相同的节点，因此需要对表格的结构进行判断，在此基础上采用默认主键和自定义主键两种形式来解决节点的唯一性，进而保证后期的节点查询可靠性。

（3）保留原数据库中约束信息

针对关系型数据中的属性类型多样性，以及 Not Null，Unique 等多种情况，使用特定映射机制和额外的用户约束信息字段等形式，确保数据在转存过程中不丢失。

2 节点与关系的转换规则

2.1 节点及其字段类型的映射

在使用关系型数据库进行数据库应用开发的过程中，通常会在前期定义好实体的属性，一个实体表即对应一个实体，从而可以自然地将实体映射为图数据库中的一个节点类型。实体包含一个或多个属性，这些属性具有多种类型如字符串、整数、布尔型等，这些特性也同样需要映射到节点类型中。同时数据库表常常使用一个主键（primary key）对每一条数据进行标识，方便后期的增删改查。因此在本文中，主要通过特定的转换规则解决上述问题，保证数据转换前后的完整性，同时在图数据库中增加额外的索引以提供额外的查询能力。

首先需要解决属性类型的映射问题，通过对关系型数据库 MySQL 中存在的属性类型进行统计分析，在最大程度保留元数据的基础上，我们定义了表 1 所列的映射规则。

表 1 数据库间属性类型的映射规则

Table 1 Mapping rules of property types between databases

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

通过表 1 中的具体映射规则，可以保证在转换前后，所有实体属性的类型可以得到最大程度的保留，因此在具体操作过程中会涉及到数据类型的强转，具体细节将在第 3 节举例说明。

其次需要解决在图数据库中节点的唯一标识的问题。在 MySQL 中，不同的表中都可以存在主键 id＝1 的数据，但在一个图数据库中，所有的节点属于一个命名空间，同一个命名空间中不允许存在两个唯一标识相等的节点，因此需要使用额外的字段对其进行标识。根据主键的设置，分为以下两种情况：

（1）已指定主键/复合主键。基于数据库表名的唯一性，采用“表名＋主键”的形式来对节点进行唯一标识，如使用复合主键，则需要在转换过程中进行额外的判空处理。

（2）未指定主键。通过指定特定字段的方式进行区分，但为保证数据的可靠性和唯一性，指定的字段应保证唯一性，否则图数据库仍无法对数据进行唯一性标识。

2.2 基于配置与中间表的边定义方法

在图数据库中，一条边从一个节点指向另一个节点，表现为：源节点—关系—目的节点。该关系的类型名称也必须是唯一的，因此在转换的过程中也需要保证关系类型的唯一性。

在对关系型数据库进行设计的过程中，有多种实体关系表示方法，如中间表和特定字段指定等，基于此本文提出了基于配置和中间表的边定义方法（Edge Definition Method based on Configuration and Intermediate Table，EDCIT）。在字段配置方法中，通过对实体的属性进行配置，将该实体的某一属性指向其他实体的属性，从而实现关系的定义；在基于中间表的方法中，通过配置中间表的字段和对应的两个实体，结合实体的主键，进而实现关系的定义。

2.2.1 基于字段配置的边定义

基于字段的配置可以让转换过程中的操作更细粒度，转换后图的结构和数据更符合用户的直观预期。但这也要求较多的人工干预，各个数据表的结构也需要更为整齐。

以一个 Person 表和 College 表为例，在默认所有表字段都会转变成 Person 实体的属性的基础上，用户可以选择某些特定字段为其设置一个额外的关系属性。如将 Person 表的 College 属性关联 College 表的 Name 属性，则在转换的过程中，可以使用 belongTo 等任意名称表示两个实体之间存在的关系。但正如之前所述，这对数据表的结构要求较高，即不能存在同一名称的 College。转换前后的数据对比如图 1 所示。

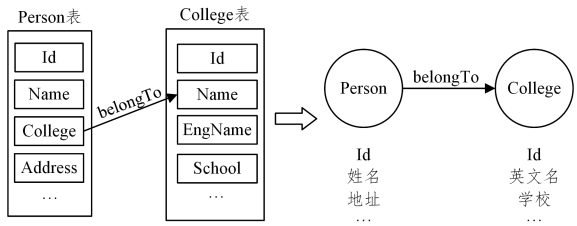

图 1 基于配置的边定义模式的映射

Fig. 1 Mapping based on configured edge definition mode

2.2.2 基于中间表的边定义

较多关系型数据库 MySQL 会使用中间表的方式建立两表之间的关系，这类表格通常会使用复合主键的形式，规定两个字段对应其他两个实体表中的主键。对于此类关系的映射，可以直接将该中间表的表名映射为图数据库中的一种关系类型。具体的映射过程如下。

以一个关系型数据库 $DB=\{T_1,T_2,T_3\}$ 为例，其中 T_1，

T_2 为实体表，T_1 为 person 表，包含{id, name, gender, address}的属性类型，且 id 为其主键；T_2 为 region 表，包含{id, name, continent, engName }属性，id 同样为其主键。T_3 表为 belongToRegion 表，包含{personId, regionId }两个属性，并使用这两个属性作为其复合主键。在转换的过程中，首先按照 2.1 节所述方案将所有的 T_1，T_2 表中的数据转换为节点添加至图数据库中，在导入所有节点后进行关系的增加。对于 $R_i \in T_3$，可以将其表示为{N_{i1}, N_{i2}}，其中 N_{i1} 为 personId 的值，N_{i2} 为 regionId 的值。在图数据库中分别寻找 id 为“person-N_{i1}”和“region-N_{i2}”的两个节点，在两节点间建立 belongToRegion 关系即可。

2.3 转存额外的约束信息

在关系型数据库 MySQL 中，会存在一些额外的字段约束如 Unique, Auto inc, Not null 等，本文通过对 MySQL 中的约束类型进行统计分析，结合使用的图数据库类型，设计了以下的转换规则，进而对这些字段的约束信息进行了保留。

NOT NULL：该约束字段规定了实体表中该字段不能为空。在图数据库中建立一个节点类型时，可以规定某些字段为非空字段，使用式(1)即可创建一个节点类型使其 C 属性不可为空。

$$Schema.vertexLabel.properties(\text{‘}A\text{’},\text{‘}B\text{’},\text{‘}C\text{’}).notNull(\text{‘}C\text{’}).ifNotExist().create() \tag{1}$$

UNIQUE：该约束字段规定了实体表中该字段具有唯一性。在转换的过程中，由于已有该约束，所以可以顺利添加所有的节点，但为保证之后新增节点时不发生冲突，在图数据库中需维持对应的变量[17]，存储已有的值，未来新增节点时会对该字段进行判断，若重复则插入失败。

AUTO INC：该约束字段会使每次新增一条数据时该属性值自增。在转换后的图数据库中，通过维持特定的变量 a 的方法，存储该字段已有的最大值。在新增节点的过程中，控制变量 a 自增即可。

3 在 Hugegraph 中实现数据转换

本节将介绍在实际转换的过程中，如何将数据从关系型数据库中转换至图数据库 Hugegraph 中。图数据库 Hugegraph 的设计理念借鉴了 Janusgraph 和 Titan[18]，使用了同样的标识包括 PropertyKey, VertexLabel, EdgeLabel, Vertex, Edge 等对一个图进行描述，并使用 Gremlin 进行检索和计算，为图数据提供了很好的检索和查询计算。另一方面，由于关系型数据库中 MySQL 结构较为典型，因此本节具体的方案同样适用于其他关系型数据库。

3.1 基于字段配置的数据转换过程

在图数据库 Hugegraph 中，属性 PropertyKey 具有唯一的类型，节点类型 VertexLabel 可以包含零个或多个属性，关系类型 EdgeLabel 应指定两个节点类型分别为起始点和终点。本节将主要介绍如何建立这 3 类最基础的元数据。

首先，建立所有的节点类型，以图 1 中的 Person 表为例，默认导入所有属性的情况下，按照表 1 中的映射关系与额外名称建立所有的 ProperyKey，Person 表的所有属性也会映射为 Person 节点类型的所有属性。对于节点的标识则使用“useCustomId”即自定义 Id 的策略，通过主键和表名设置节点 Id，对于已有主键/复合主键的表，直接使用对应的字段即可。在 Person 表中如有 Id＝4 的数据，则该节点的 Id 为“person-4”，属性和节点类型的映射关系如图 2 所示。

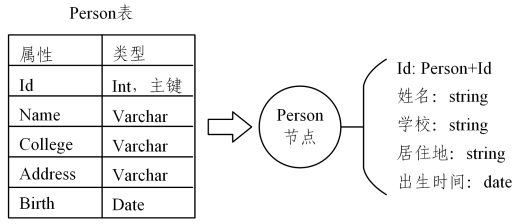

图 2 Hugegraph 中节点类型的映射

Fig. 2 Mapping of vertex types in Hugegraph

其次，基于配置实现关系类型的建立。例如，Person 表的 College 字段指向 College 表的 Name 字段，即建立一个 BelongToCollege 的关系类型，从 Person 节点指向 College 节点，并对这些信息进行暂存，便于后期插入真正的边，数据格式为：“Person. College---BelongToCollege---College. Name”。

最后，建立所有节点与关系的实例即 Vertex 与 Edge。如果在建立节点的同时建立边，会出现尚无对应节点的情况，因此必须先建立所有的节点，再根据上一步暂存的所有信息去建立关系，具体步骤如下。

(1)遍历所有数据表，建立根据表格对应的节点类型，建立所有的节点，此时不考虑 Person 表的 College 属性为一条边的情况。

(2)遍历上一步暂存的关系，以上一步的 Person 表和 College 表为例，执行下述操作：

1)遍历 Person 表的所有数据，同时暂存 College 字段所有可能的值；

2) 使用 Gremlin 语句查询所有的节点，构建一个 Map〈String, Vertex〉用于存储名称和节点的对应关系；

3) 再次遍历 Person 表的所有数据，按照第 2)步中存储的映射关系，建立所有的 BelongToCollege 关系。

(3)反复执行步骤(2)中的操作，直至建立完成所有的关系。

3.2 基于中间表的数据转换过程

在基于中间表的模式下，数据库的表格会分为实体表和中间表。因此在图数据库中建立 Schema 和具体的数据可以从这两类表入手，以下方对于实体表和中间表的配置为例，具体的转换流程如下。

```
{
  “dbName”:“xh”,
  “tables”:[
    {
      “name”:“person”,
      “type”:“entity”,
      “entityTableConfig”:{
        “primaryKey”:“id”
      }
    },
```

```
    {
      "name":"college",
      "type":"entity",
      "entityTableConfig":{
        "primaryKey":"id"
      }
    },
    {
        "name":"belongToCollege",
        "type":"middle",
        "midTableConfig":{
        "sourceTable":"person",
        "sourceTableColumn":"id",
        "targetTable":"college",
        "targetTableColumn":"id",
        "interTableSColumn":"personId",
        "interTableTColumn":"collegeId"
      }
    }
  ]
}
```

(1)建立所有的节点类型(VertexLabel)和节点(Vertex)。首先遍历所有的实体表，即 type＝entity 的所有数据表。根据配置信息中的 primaryKey 或默认的主键方式，建立对应的节点类型，同时使用 2.1 节的方案建立所有的 PropertyKey。建立了当前节点类型后，使用 SQL 语句获取实体表中的所有实体，进而建立当前的所有节点。

(2)建立所有的关系类型(EdgeLabel)和关系(Edge)。遍历所有关于中间表的配置，即 type＝middle 的所有数据表。根据其中的配置信息建立关系并进行暂存，以转换流程的中间表配置项为例，暂存“belongToCollege. personId－person. id”，“belongToCollege. collegeId－college. id”，“personId---belongToCollege－collegeId”这 3 个三元组，将这 3 个三元组进行分析映射至图数据库中，即可得到 person－belongToCollege－college 的关系类型。建立了当前的关系类型后，同样使用 SQL 查询语句，获取该中间表的所有数据，建立所有关系即可。

4 实验分析

本文实验使用的设备信息为：处理器 Core i5-8250U@1.60GHz，内存 DDR4 2 400 MHz 8 GB，系统版本为 Windows10 1903。使用 Java 作为基础开发语言，并通过在 IDE 中打点的方法对转换时间进行统计分析。

通过对不同量级的数据集进行实验，验证了本文提出的将数据从关系型数据库转存至图数据库 Hugegraph 的方案的可行性，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，使用了 MySQL 官方提供的 world 和 sakila 两个数据集，分别对第 3 节提出的两种转换模式进行验证。

4.1 基于字段配置的边定义方案的验证

本实验验证基于 MySQL 官方提供的 world 数据集，这是一个统计了全球所有国家和城市的数据库，包含 city，country，countrylanguage 共 3 个数据表。

按照 3.1 节提出的方案，将 city 表中的 CountryCode 属性与 Country 表的 Code 属性关联，进而建立“属于…国家”的关系。通过使用逐个插入数据和批量插入数据的方案分别进行测试，结果如下：共 239 个国家，4 079 个城市，根据配置建立图数据库中的 Schema 耗时 1.09 s，逐个插入节点耗时 7.69 s，逐个建立关系耗时 37.3 s，批量插入节点耗时 1.05 s，批量插入关系耗时 15.7 s。

数据转换完成后，需要对数据的可靠性进行验证，根据一些常见的业务场景，我们对表 2 列出的语句进行了实验验证并记录其返回值，结果如表 2 所列。

表 2 字段配置方法的结果验证

Table 2 Result verification of field configuration method

业务场景	Gremlin 语句	返回值
查询全球国家个数	g. V(). hasLabel('country'). count()	[239]
查询全球城市个数	g. V(). hasLabel('city'). count()	[4079]
加拿大包含哪些城市	g. V(). hasLabel('country'). has('name','Canada'). in('属于…国家')…	[city-1810，city-1811，…city-1858]
Cambridge 属于哪个国家	g. V(). hasLabel('city'). has('name','Cambridge'). out('属于…国家')…	[country-CAN]

4.2 基于中间表的边定义方案的验证

本实验验证基于 MySQL 官方提供的 sakila 数据集，这是一个模拟 DVD 租赁信息管理的数据库，包含了 actor，film，staff，store 等多个实体表用于描述 DVD、员工以及仓库的相关信息，以及 film_actor，film_category，rental 等多个中间表用于描述电影与演员的关系、租借的记录等。

通过对不同数量级的数据条数进行转换测试，对耗时进行统计，结果如表 3 所列。

表 3 转换耗时统计

Table 3 Conversion time statistics

vertexes count	edges count	mapping time/s	data insert (single)/s	data insert (batch)/s
1 200	5 462	1.492	20.588	1.457
5 000	23 952	1.855	75.128	4.231
20 000	76 521	1.692	320.452	15.318

由表 3 可以看出，整体耗时与转换数据量近乎呈等比例增长，且主要的耗时在于对图数据库 Hugegraph 的节点/关系的插入：在默认单条逐个添加节点的情况下，单位时间可以插入节点/关系大约 350 个；在通过提前缓存数据，最终进行一次性插入后，插入数据耗时大幅降低，单位时间可以插入节点/关系大约 5 000 个。

数据转换完成后，同样需要对该数据集的可靠性进行验证，根据一些常见的业务场景，我们对于 1 200 个节点的数据集使用表 4 中的语句进行了实验验证并记录了其返回值，结果如表 4 所列。

表 4 中间表方法的结果验证

Table 4 Result verification of intermediate table method

业务场景	Gremlin 语句	返回值
查询所有 film 个数	g. V(). hasLabel('film'). count()	[1000]
随机查询 3 个电影	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']

由表 4 可以看出，转换后的数据具有较强的可靠性，通过与最新的工作对比可以发现，该转换方法的适用场景更广，通过简单的修改即可适用于其他类型的图数据库，在实验测试机器上转换效率每秒可以达到 5000 个节点或边，结合使用的图数据库 Hugegraph，可以实现数据的快速转换。

结束语 本文设计了一种数据转换方法对关系型数据库中的数据进行转换，实现了节点和关系的映射，并最大程度地保留了原数据库的约束，最终将数据成功导入图数据库 Hugegraph 中。由于当前学术界对于此类转换方法的研究较少，且大都局限在特定的数据集或特定的数据库如 Neo4j，而本文提出的映射方法具有通用性，即适用于基于 TinkerPop 3 框架的多类图数据库后端，具有一定的独创性。但本文方法仍需进一步改进，如支持对外键的配置以及关系型数据库的更丰富的特性，以扩大数据源的适配面。

参 考 文 献

[1] IAN R, JIM W, EIFREM E. Graph Databases [M]. O'Reilly Media, Inc. : Cambridge, 2015: 12-20.

[2] NEEDHAM M, HODLER A E. Graph Algorithms[M]. O'Reilly Media, Inc. : California, 2019: 5-8.

[3] PAUL S, MITRA A, KONER C. A Review on Graph Database and its Representation[C]// International Conference on Recent Advances in Energy-efficient Computing and Communication. 2019: 1-5.

[4] ZHAO P, SHOU L D, CHEN K, et al. Storage and Query Model for Localized Search on Temporal Graph Data[J]. Computer Science, 2019, 46(10): 186-194.

[5] NEO4J STAFF. The Database Model Showdown: An RDBMS vs. Graph Comparison[EB/OL]. (2015-08-03) [2020-11-05]. 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[C]// Conference Proceedings By Track. 2017: 156-165.

[7] MAGORZATA P W, RYKOWSKI D. Comparison of Relational, Document and Graph Databases in the Context of the Web Application Development[C]// Information Systems Architecture and Technology: Proceedings of 36th International Conference on Information Systems Architecture and Technology. Springer International Publishing, 2016: 3-13.

[8] FOSIC I, ŠOLIC K. Graph database approach for data storing, presentation and manipulation[C]// 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics. 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). 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: 509-512.

[11] MUELLER W, IDZIASZEK P, GIERZ U, el al. 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. 2019.

[12] UNAL Y, OGUZTUZUN H. Migration of data from relational database to graph database[C]// the 8th International Conference. 2018: 1-5.

[13] DE VIRGILIO R, MACCIONI A, TORLONE R. R2G: a Tool for Migrating Relations to Graphs[C]// International Conference on Extending Database Technology. 2014: 640-643.

[14] DE VIRGILIO R, MACCIONI A, TORLONE R. Converting relational to graph databases[C]// International Workshop on Graph Data Management Experiences and Systems. 2013: 1-6.

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

[16] SERIN F, METE S, GUL M, et al. Mapping between relational database management systems and graph database for public transportation network[C]// International Research/Expert Conference. 2018: 209-212.

[17] linlin1989117. HugeGraph 之 Variables[EB/OL]. (2020-10-14) [2020-11-7]. https://blog. csdn. net/linlin1989117/article/details/109072676.

[18] thutmose. "JanusGraph 与 HugeGraph"图形数据库-技术选型-功能对比[EB/OL]. (2019-03-25) [2020-11-07]. https://blog. csdn. net/lovebyz/article/details/88800363.

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.