doi:10.12068/j.issn.1005－3026.2025.20230298

面向双时态RDF星型查询的两阶段索引方法

王 刚，张 薇，陈宏栉，张 富
（东北大学 计算机科学与工程学院，辽宁 沈阳 110169）

摘 要：为了对时间信息进行表示和处理，随之开展了时态RDF（temporal resource description framework）的扩展研究，而如何对时态RDF数据进行有效的管理也逐渐成为了一个重要问题．构建有效的索引机制是管理时态RDF数据和提高查询效率的重要途径之一．本文提出了一种针对双时态RDF（bitemporal RDF）星型查询的两阶段索引方法：第一阶段索引提出一种基于双时态RDF节点邻域信息的候选集生成方法，第二阶段索引进一步提出一种基于位图索引的结果集生成方法．通过两阶段索引可以避免星型查询中成本昂贵的连接操作，快速得到查询结果．在查询效率和索引性能方面进行了对比实验，验证了所提方法的有效性．

关 键 词：资源描述框架（RDF）；双时态RDF；星型查询；两阶段索引；位图索引
中图分类号：TP 18 **文献标志码**：A **文章编号**：1005-3026(2025)02-0018-10

Two-Stage Indexing Method for Bitemporal RDF Star Query

WANG Gang, *ZHANG Wei*, *CHEN Hong-zhi*, *ZHANG Fu*
（School of Computer Science & Engineering, Northeastern University, Shenyang 110169, China. Corresponding author: ZHANG Fu, E-mail: zhangfu@mail.neu.edu.cn）

Abstract: In order to represent and process time information, the extended research on temporal RDF (temporal resource description framework) model has been carried out, and how to effectively manage temporal RDF data has gradually become an important issue. Constructing a reasonable and effective index mechanism is one of the important ways to manage temporal RDF data and improve query efficiency. A two-stage indexing method for bitemporal RDF star query was proposed. First-stage indexing proposes a candidate set generation method based on bitemporal RDF node neighborhood information, and second-stage indexing further proposes a result set generation method based on bitmap indexing. With two-stage indexing, expensive join operations in star queries can be avoided and query results can be obtained quickly. This paper conducts comparative experiments in terms of query efficiency and index performance to verify the effectiveness of the proposed method.

Key words: resource description framework (RDF); bitemporal RDF; star query; two-stage indexing; bitmap indexing

资源描述框架（RDF）及其标准查询语言SPARQL是用于表示和查询万维网上资源信息的标准语言[1]．RDF凭借其灵活简单的结构（即<主语、谓语、宾语>的三元组结构）以及可共享等特性被应用在越来越多的领域中，例如知识图谱和自然语言处理等．当前RDF已经成为许多领域进行知识表示和处理的重要技术之一．

标准RDF描述的信息是静态的，不会随时间变化．而时间信息作为事物的固有属性存在于许多应用领域中．为了使RDF模型能够对动态资源进行描述，研究者们在标准RDF的基础上提出了时态RDF模型[2]的概念．时态RDF模型通过引入时间概念，使得RDF模型具备对动态资源进行描述的能力．时态RDF模型可以记录事物的发展变化，对数据进行版本管理等．特别是，有效时间（valid time）和事务时间（transaction time）作为数

收稿日期：2023-10-27
基金项目：国家自然科学基金资助项目（62276057）．
作者简介：王 刚（1978—），男，辽宁本溪人，东北大学讲师；张 富（1983—），男，内蒙古呼和浩特人，东北大学教授，博士生导师．

据库和许多领域最主要的2种时态数据类型,研究者逐渐研发了基于有效时间或事务时间扩展的单时态RDF模型,包括tRDF[3]、Temporal RDF[4]和TemRDF[5]等.

随着对时态RDF模型的研究和应用不断深入和扩展,时态RDF数据的规模也在不断地扩大.如何对时态RDF数据进行有效的管理成为了一个重要问题.文献[6]提出了一种基于HBase的时态RDF的存储和查询方法.文献[7-8]分别提出了用于查询RDF事务时间和有效时间的SPARQL时态扩展查询语言LSPARQL和SPARQLMT,并给出了具体的查询语法和语义定义.文献[9]给出了一种用于表示和查询双时态数据的RDF扩展形式Bi-VAKs.文献[10]提出了一种用于表示和查询有效时间的RDF扩展模型,同时给出了基于关系数据库的时态RDF的存储方法.上述工作主要集中在如何扩展RDF和SPARQL实现时态数据的表示、查询以及管理等问题.构建合理有效的索引机制也是管理时态RDF数据和提高查询效率的重要途径之一.然而,目前有关时态RDF索引的相关技术研究还很有限,大部分的索引研究都是基于标准RDF模型[11]开展的.近几年,文献[12]提出了一种处理包含事务时间数据的RDF图索引结构,该索引使用KD树(K-dimension tree)对时间信息进行索引,将数据按时间信息进行分组,之后使用位图索引来对RDF三元组数据进行匹配.文献[13]提出一种将主语和宾语作为键值和时间信息相关联来存储和检索时间信息的查询索引.文献[14]提出将时态边视为图的日志,并构建快照以进行查询优化.有关上述提到的时态RDF表示模型和索引技术的详细介绍,可参见文献[15].

上述针对时态RDF索引的研究成果主要是针对单时态RDF构建的索引.目前,支持双时态(即同时支持有效时间和事务时间)RDF模型的索引技术非常少.有效时间和事务时间是实际应用领域中广泛存在的2种时态数据类型,有关双时态RDF的建模和查询需求将进一步增强标准RDF的表达能力并拓宽RDF的应用范围.研究支持双时态RDF模型的索引技术将进一步提升时态RDF数据的查询效率,成为RDF和许多领域亟待解决的重要问题.

为此,本文考虑到星型查询(star query)是实际场景中使用非常频繁和非常重要的查询类型[16],针对星型查询和双时态RDF数据的特点,提出了针对双时态RDF(bitemporal RDF)星型查询的两阶段索引方法.第一阶段索引提出一种基于双时态RDF节点邻域信息和B+树的候选集生成方法,第二阶段索引进一步提出一种基于主语、谓语位图索引的结果集筛选和生成方法.通过两阶段索引可以避免星型查询中成本昂贵的连接操作,快速得到查询结果.本文构建了双时态RDF数据集,并在查询效率和索引性能方面进行了对比实验,实验结果验证了所提方法的有效性.

1　双时态RDF模型

常见的时间信息主要包括有效时间和事务时间.有效时间表示了信息在真实世界中有效的期限,而事务时间表示了信息被插入到数据库中的时间.本文在标准RDF[1]基础上,进一步给出了双时态RDF模型的形式化定义,在双时态RDF模型的基础上进行后续的索引研究.

定义 1(双时态RDF模型)　双时态RDF模型是由同时具有有效时间和事务时间标签的RDF三元组组成.形式上,双时态RDF三元组可以表示为$(s, p, o):[t_{v,s}, t_{v,e}][t_t]$,其中:

1)s,p,o分别表示标准RDF三元组中的主语(subject)、谓语(predicate)和宾语(object).

2)$[t_{v,s}, t_{v,e}]$代表RDF三元组的有效时间,该有效时间从时间点$t_{v,s}$开始到时间点$t_{v,e}$结束,$t_{v,s} \leqslant t_{v,e}$.当$t_{v,s}=t_{v,e}$时,表示此三元组的有效时间为1个时间点.

3)t_t代表RDF三元组的事务时间.即三元组被插入到数据库中的时间.

表1给出了1个双时态RDF数据的部分示例,其中rdf:type代表主语的类型.可以看出,三元组同时包含了有效时间和事务时间,这使得查询中可能涉及复杂的时态查询.在查询中,1个实例的相关信息是分散在多个三元组中的,通过不同的谓语与对应的宾语相关联从而构建出主语实例的全部信息.因此,查询某个具体的实例需要通过多个查询语句加以限制才能确定,这种情况下使用的查询类型就是星型查询.星型查询是RDF中使用较为频繁的查询类型[16],是很多复杂查询的组成部分.

图1展示了查询表1中数据的1个星型查询示例.在该图中,Prefix定义了查询中所使用的命名空间的缩写;select声明了需要查询的变量;

where里定义了2条查询子句,这2条查询子句具有相同的主语变量"?x",进而构成了1个星型查询,其中第1条查询子句通过rdf:type限定了查询变量的类型是org:FullProfessor,且满足有效时间t_v和事务时间t_t的双时态约束,第2条查询子句限定了查询变量所在的工作部门.

表1 双时态RDF数据示例

Table 1 An example of bitemporal RDF data

主语	谓语	宾语	有效时间	事务时间
dep:*S*0	rdf:type	org:FullProfessor	2000-12—2010-07	2000-12
dep:*S*0	org:teacherOf	dep:Course0	2000-12—2010-07	2000-12
dep:*S*0	org:worksFor	Department0	2000-12—2010-07	2000-12
dep:*S*1	rdf:type	org:FullProfessor	2010-09—now	2011-11
dep:*L*0	rdf:type	org:Lecturer	2002-03—2010-06	2012-12
⋮	⋮	⋮	⋮	⋮

```
Prefix org=<http://example.org/uni#>
Prefix rdf=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?x
where {
  ?x rdf:type org:FullProfessor 2001.3<vt<2010.12 tt>2000.9.
  ?x org:worksFor <http://www.Department0.University0.edu> .
}
```

图1 星型查询示例

Fig. 1 An example of star query

2 两阶段索引结构

2.1 索引整体结构

根据星型查询的特点,星型查询中涉及的连接类型主要是主语与主语之间的连接.查询越复杂,查询的语句越多,查询过程中需要进行的连接操作就越多.因此,从避免连接操作的角度出发,设计了针对双时态RDF数据星型查询的两阶段索引.同时,将索引设计为两阶段主要是针对双时态RDF数据,既包含了三元组信息又包含了双时态信息的特点.在索引的第一阶段主要关注RDF三元组信息的处理,在第二阶段进一步处理时态信息.

在处理查询时,索引的第一阶段通过查询中给定的宾语信息获得1个候选集;第二阶段对候选集进行进一步验证并产生最后的结果集,从而避免成本昂贵的连接操作.从查询中给定的宾语信息出发可以快速缩小查询的范围,同时减小第二阶段需要验证的数据规模.索引的总体结构图如图2所示,其中主语、谓语和宾语分别用S,P和O表示,V表示构建的B+树中的节点.第一阶段提取RDF图中的邻域信息,这里的邻域信息指RDF图中每个主语节点周围的宾语节点信息以及每个宾语节点周围的主语节点信息.通过遍历数据集提取邻域信息并将这些信息存储在B+树中.在处理查询时,提取查询中的信息并与第一阶段获得的邻域信息进行比较,得到1个候选集;第二阶段利用已有的候选集和谓语信息,使用主语和谓语的位图索引来进行验证.如果查询中不涉及时态信息,则对宾语信息进行验证;如果查询中涉及时态信息,则在对宾语信息进行验证的同时对时态信息进行验证或提取.

2.2 第一阶段索引结构

第一阶段索引的目的是通过查询中给定的宾语信息生成1个候选集.本文提出了1种基于双时态RDF节点邻域信息和B+树的候选集生成方法.第一阶段索引提取了RDF数据中的所有主语S和宾语O的关系.针对每个S分别记录与其相连的所有O的值,并记录其中的最大值和最小值.针对每个O分别记录与其相关联的所有S的值.通过第一阶段索引,从RDF图中提取了节点的邻域信息.RDF图是RDF数据的最自然的表达形式.在RDF图中,数据表示为由节点和边组成的有向图.每个节点表示1个资源,连接节点的边表示2个资源之间的关系.数据集规模变大时,各个资源相互连接,构成了更复杂的RDF图.RDF图蕴含了RDF数据的所有信息,但将RDF数据以关系表的形式存储于关系型数据库时很难利用RDF图的信息.关系型数据库经过长时间的发展,技术更加成熟稳定且更具有普遍性.将数据存储于关系型数据库中,依旧是目前的主流解决办法.因此通过将RDF图的邻域信息保存在索引中,一方面,解决了关系型数据库无法利用RDF图信息的缺点;另一方面,利用RDF图信息可以更加快速有效地过滤数据,减少关系型数据库中成本昂贵的连接操作的发生.利用RDF图中的邻

域信息,可以将查询范围限制在现有条件中的宾语和主语的邻域信息内,可以用最小的查询成本代价过滤掉大部分数据,获得候选集.通过这种方式可以使索引具有很好的可扩展性,在数据规模扩大时,依旧可以获得很高的查询效率.

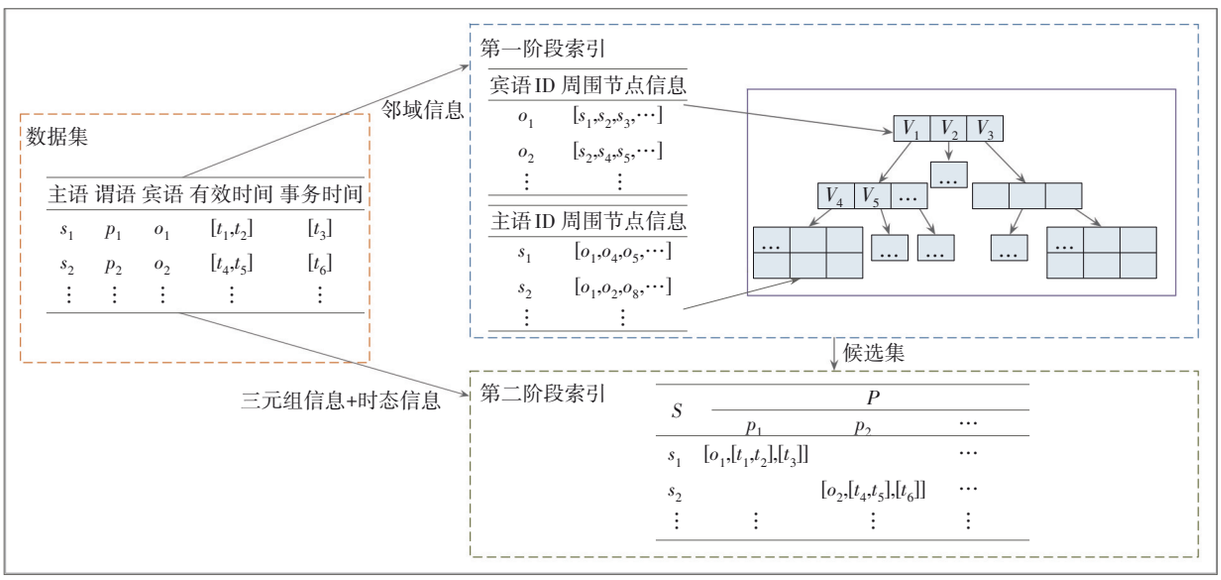

图2　两阶段索引的总体结构图

Fig. 2　Overall structure diagram of the two-stage index

在构建第一阶段索引时,以每个宾语O的值作为B+树的键值,B+树中值的位置分别存储了与键值相关联的所有主语的信息.每个主语S中又相应存储了与其相关联的所有宾语O的值.在查询时,从查询条件中找到具有最少关系数的宾语值进行查找.通过查找可以获得与此宾语值相关联的所有S值.针对获得的S值,将查询中所有涉及的宾语值与其进行匹配.如果与此S相关联的所有宾语值中包含了查询中涉及的全部的宾语值,则将此S值加入到候选集中.具体构建第一阶段索引的过程主要分为以下几步.首先,遍历数据集,分别为数据集中的主语、谓语和宾语分配1个整数ID值.因为主语和宾语在RDF图中都表示为实体节点.某些元组中的主语同时也可能是某些元组中的宾语,因此将主语和宾语统一进行ID值的分配.其次,在遍历过程中针对每一个宾语O记录与其所有相关联的S的值.同时,针对每一个S记录所有与其相关联的O的值,并记录其中的最大值和最小值.最后,以O的ID值为键,并以与O相关联的S的ID值以及相关信息为值构建B+树.图3为表1中的双时态RDF数据的RDF图表示形式.因为在第一阶段索引中不处理时态信息,所以在图3中只展示了三元组数据部分,其中type(即rdf:type)代表主语的类型,不带圆圈的数字表示三元组中主语或宾语对应的ID值,带圆圈的数字表示谓语的ID值.

根据图3中的信息,针对每个宾语和主语分别提取其邻域信息,并构建对应关系.最后得到的宾语和主语的邻域信息如表2所示.第一阶段索引使用B+树来对数据进行存储和检索,如图4所示.构建的B+树以表中的宾语ID为键,以对应的主语ID、最小值MIN、最大值MAX和主语的邻域信息共同组成的对象为值.主语的邻域信息以数组的形式存储于B+树的叶子节点中.宾语的邻域信息则以键值对的形式隐式存储于B+树中,查询时以关系数最小的宾语值作为查询入口.

2.3　第二阶段索引结构

第二阶段索引的目的是对候选集进行进一步验证筛选并产生最后的结果集.本文提出一种基于主语和谓语位图索引的结果集筛选和生成方法.与标准RDF构建的位图索引不同,这里构建的位图索引中除了存放对应的宾语信息,同时也存放了对应的有效时间和事务时间的信息.因为在第一阶段索引中,已经对主语和宾语信息的值进行了验证,加入到候选集中的主语和宾语值间的关系已经满足了查询条件,第二阶段的目标是对候选集中主语的谓语信息和时态信息是否符合查询条件进行验证.根据主语和谓语的值,首先通过位图索引验证查询条件中已知的宾语值是否匹配.同时,如果查询条件中对时态信息

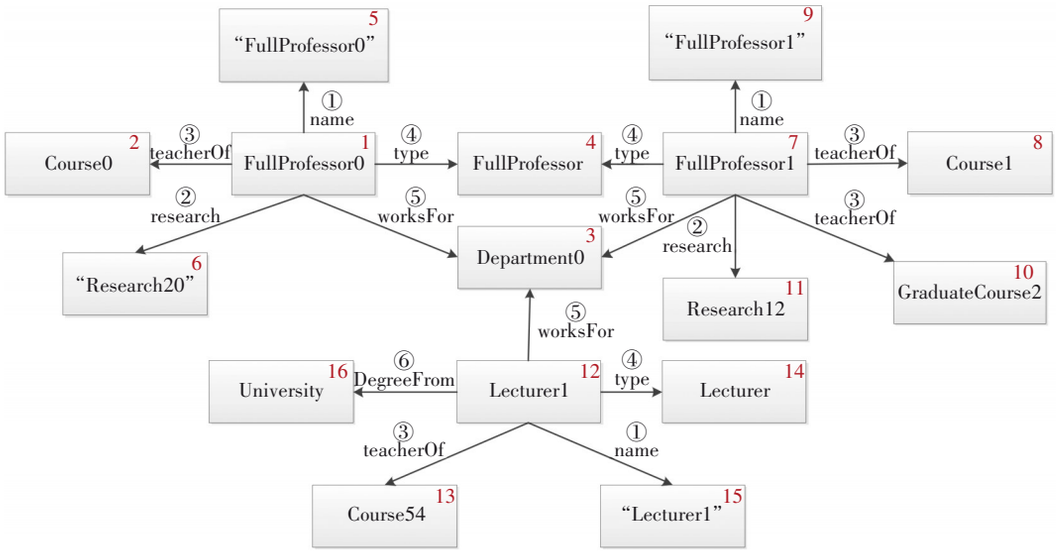

图 3 表 1 中数据的 RDF 图

Fig. 3 RDF graph of the data in table 1

表 2 通过第一阶段索引构建的 RDF 数据形式

Table 2 RDF data forms constructed with first-stage index

宾语	主语	(最小值,最大值)	主语的邻域信息
2	1	(2,6)	[2~6]
3	1	(2,6)	[2~6]
	7	(3,11)	[3,4,8~11]
	12	(3,16)	[3,13~16]
4	1	(2,6)	[2~6]
	7	(3,11)	[3,4,8~11]
5	1	(2,6)	[2~6]
6	1	(2,6)	[2~6]
8	7	(3,11)	[3,4,8~11]
9	7	(3,11)	[3,4,8~11]
10	7	(3,11)	[3,4,8~11]
11	7	(3,11)	[3,4,8~11]
13	12	(3,16)	[3,13~16]
14	12	(3,16)	[3,13~16]
15	12	(3,16)	[3,13~16]
16	12	(3,16)	[3,13~16]

进行了约束(例如图 1 中的 $t_t > 2000\text{-}09$),则需要验证查询结果中的时态信息是否满足该约束.如果通过了验证且查询结果中有需要进行查询的宾语变量或时态信息,则进一步获得对应的宾语值和时态信息作为结果与主语信息一同进行返回.根据表 1 中的数据及其 ID 值构建的位图索引如表 3 所示.

3 索引维护算法

双时态 RDF 数据由于包含了双时态信息,其频繁的更新操作对索引的可维护性提出了更高的要求.为此,本节在上述两阶段索引的基础上,进一步研究并给出了两阶段索引的维护算法,主要包括索引的新增和删除算法.

在双时态 RDF 数据更新时,需要按照第 2 节提到的方法对邻域信息进行同步更新.在此基础上,算法 1 和 2 分别给出了第一阶段和第二阶段索引数据的新增过程.

算法 1 第一阶段索引的新增

输入:*S*_set, *s*_id, *o*_id, bPlusTree //其中 *S*_set 是所有主语的集合,*s*_id 是主语对应的唯一标识符(即 ID 值),*o*_id 是宾语对应的 ID 值,bPlusTree 是原第一阶段索引

输出:bPlusTree //新增的第一阶段索引

```
1   s_info = S_set.get(s_id) //根据主语的 ID 值从 S_set 中获得主语的邻域节点的信息
2   node = bPlusTree.search(o_id) //利用宾语的 ID 值在第一阶段索引中搜索
3   if (node = = null){  //如果不能找到对应的节点
4       bPlustTree.insert(o_id, s_info)) //则在第一阶段索引中插入新的节点
5   }
6   else{  //如果能根据宾语的 ID 值在第一阶段索引中找到对应的节点
7       if (node.search(s_id)! = null){
8           node.delete(s_id)
9   }
```

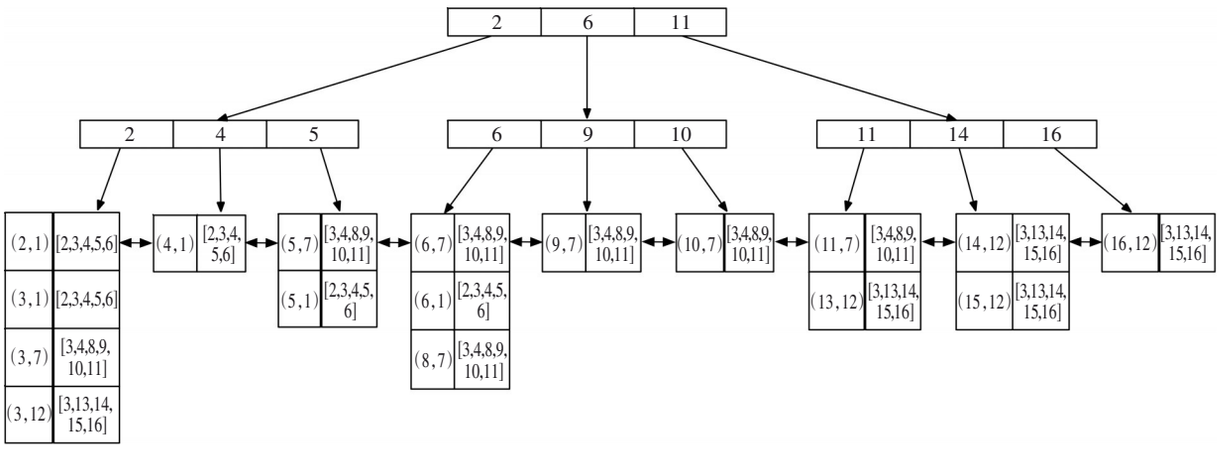

图4　第一阶段索引中构建的B+树

Fig. 4　The B+ tree constructed in first-stage indexing

表3　通过第二阶段索引构建的主语、谓语位图索引

Table 3　Subject predicate bitmap constructed with second-stage indexing

S	*P*					
	1	2	3	4	5	6
1	5 [2000-12—2010-07] [2000-12]	6 [2000-12—2005-09] [2000-12]	2 [2000-12—2010-07] [2000-12]	4 [2000-12—2010-07] [2000-12]	3 [2000-12—2010-07] [2000-12]	
7	9 [2000-12—2010-07] [2000-12]	11 [2000-12—2010-07] [2000-12]	{8, [2000-12—2010-07] [2000-12]}, {10, [2000-12—2010-07] [2000-12]}	4 [2000-12—2010-07] [2000-12]	3 [2000-12—2010-07] [2000-12]	
12	15 [2000-12—2010-07] [2000-12]		13 [2000-12—2010-07] [2000-12]	14 [2000-12—2010-07] [2000-12]	3 [2000-12—2010-07] [2000-12]	16 [2000-12—2010-07] [2000-12]

10　　node.add(*s*_info) //将新增数据的信息添加到节点中

11　}

12　return bPlusTree

对第一阶段索引进行新增,第一步根据主语对应的整数ID值从主语集合*S*_set中获得新增数据主语的周围节点的信息*s*_info(第1行).第二步,利用宾语的ID值在第一阶段索引中进行搜索(第2行).第三步,如果发现根据宾语的ID值不能在索引中找到对应的节点,则在索引中以宾语的ID值*o*_id为键,以*s*_info为值在第一阶段索引中插入新的节点(第3~5行).第四步,如果根据宾语的ID值获得了对应的节点,则在节点中进一步根据主语的ID值检查在节点中是否已经存在与此次新增的主语的相关邻域信息.如果相关信息已经存在,则将其在节点中进行删除.如果相关信息不存在或已经将原信息进行了删除,则再将新的相关信息在节点中进行添加即可(第6~11行).最后,返回新增算法更新后的第一阶段索引(第12行).

算法2 第二阶段索引的新增

输入:*s*_id, *p*_id, *o*_id, t_v, t_t, spBit //*s*_id, *p*_id, *o*_id分别表示新增数据的主语、谓语、宾语的ID值,t_v, t_t表示有效时间和事务时间的值,spBit是原第二阶段索引

输出:spBit //新增的第二阶段索引

1　if $\left(s\_\mathrm{id} > \mathrm{spBit}.m \| p\_\mathrm{id} > \mathrm{spBit}.n\right)$ { //如果新增数据的ID值大于第二阶段位图索引的现有长度

2　　spBit ← reBuildSPBit() //则需要重新构建索引

```
3   }
4   else{  //否则就在原索引的基础上进行数据的添加
5       if(spBit[s_id][p_id]) == =null{  //如果当前主语谓语ID锁定的位置为空
6           spBit[s_id][p_id] = new data(o_id, t_v, t_t) //则新建1个数据对象进行添加
7   }
8       else{  //如果当前主语谓语锁定的位置不为空
9       spBit[s_id][p_id].add(o_id, t_v, t_t) //在原数据基础上添加当前数据
10      }
11  }
12  return spBit
```

如算法2所述,第二阶段索引的新增需要检查新增数据的主语和谓语ID整数值是否大于第二阶段位图索引的长度.如果新增数据的ID值大于位图索引的现有长度,则需要利用更新后的邻域信息重新构建第二阶段位图索引(第1~3行).如果新增数据的主语和谓语的ID值在当前位图索引长度的范围内,则可以在原索引的基础上新增数据.根据新增主语、谓语的ID值在位图索引中进行查找.如果锁定的位置为空,则新建1个对象并添加到对应的位置(第4~7行).如果锁定的位置不为空,则说明此主语和谓语关系不只存在1个相对应的宾语,就在锁定的位置将宾语信息、有效时间和事务时间进行新增(第8~10行).最后同样将新增数据后的第二阶段位图索引返回(第12行).

对于索引的删除,考虑到第一阶段索引和第二阶段索引的删除过程类似,下面算法3直接给出了第一阶段索引和第二阶段索引的删除过程.对于第一阶段索引的删除,首先是根据待删除数据的宾语的ID值找到对应的节点.在节点中将主语的ID值删除(第1~2行).如果删除数据后,发现节点为空,则进一步将节点在第一阶段索引中删除(第3~5行).对于第二阶段索引的删除,则是根据主语和谓语的ID值锁定到对应的位置,将对应位置上的宾语的ID值删除.同时也将宾语ID值对应的有效时间和事务时间信息一同删除(第6行).通过以上步骤,就完成了第一阶段索引和第二阶段索引的删除.最后将第一阶段索引和第二阶段索引返回(第7行).

算法3 第一阶段与第二阶段索引删除

输入:s_id, p_id, o_id, bPlusTree, spBit //s_id, p_id, o_id分别表示待删除数据的主语、谓语、宾语ID值,bPlusTree是原第一阶段索引,spBit是原第二阶段索引

输出:bPlusTree, spBit //删除数据后的第一阶段索引和第二阶段索引

```
1   node = bPlusTree.search(o_id) //根据宾语ID值在第一阶段索引中找到对应的节点
2   node.delete(s_id) //在节点中将主语ID值删除
3   if(node = = null) {  //如果删除数据后节点为空
4       bPlusTree.delete(node)//则将节点也一并删除
5   }
6   spBit[s_id][p_id].delete(o_id) //根据主语和谓语ID值在第二阶段索引中删除对应位置的值
7   return bPlusTree, spBit
```

4 实验评估

4.1 实验环境及数据集

本实验使用的编程语言是Java,使用的开发工具是IntelliJ IDEA,使用的数据库是MySql.本实验是在具有Intel i5-10300H处理器,16 GB运行内存的Windows10操作系统的计算机上完成的.

由于目前尚未存在公开的时态RDF标准数据集,本实验使用的数据集是在经典RDF评价数据集LUBM[17]的基础上构建而成的.LUBM是经典的RDF推理、存储和查询系统的评价数据集之一.考虑到原始的LUBM数据集是不包含时间信息的,因此本文通过在基础的LUBM数据集上增加双时态信息构建了LUBM双时态RDF数据集.通过在每条LUBM数据集上添加1个时间区间作为有效时间,1个时间点作为事务时间,构建了实验所需的数据集.为了构建的数据集更加贴近真实情况数据集,时间点以及时间区间以随机的方式进行添加,并且为了模拟现实世界中存在的时间重叠和重合的情况,添加的时间区间数和时

间点数小于三元组的数量.数据集的规模和时间信息的规模如表4所示.

在上述数据集中,不同主语的数量分别为2 408,9 875,23 992和48 218.因此,在数据集中每个主语平均会出现5~6次,这使得三元组之间的联系相对复杂,查询也可能相对复杂.

表4　双时态RDF数据集

Table 4　Bitemporal RDF datasets

数据集	双时态三元组数	时间区间数	时间点数
数据集1	16 000	5 000	2 500
数据集2	50 000	8 000	4 000
数据集3	140 000	16 000	8 000
数据集4	300 000	32 000	16 000

在对比实验上,考虑到本文在前面提到的已有其他工作大多数都主要集中在如何扩展RDF进行时态数据的表示、查询和存储等问题,并不是专门针对有效时间或事务时间提出的索引,而文献[12]提出的基于KD树的索引是针对带有事务时间的RDF索引.为此,本文选择与文献[12]进行了对比实验.需要说明的是,由于文献[12]提出的索引方法仅适用于单一时态的事务时间,因此在实验中,仅在查询条件包含事务时间的情况下,与其进行了对比实验.

4.2　查询效率实验分析

为了验证所提索引的查询效率,本文针对常见的查询形式设计了不同种类和查询难度的星型查询,在实验中针对每类查询取5次实验的平均值作为最后的结果.首先,图5中的StarQuery1与图1中的查询示例类似,但该查询没有涉及对时态信息的查询和约束,图6给出了本文提出的两阶段索引在StarQuery1上的查询性能表现.

```
Prefix org=<http://example.org/uni#>
Prefix rdf=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?x
where {
  ?x rdf:type org:AssociateProfessor.
  ?x org:worksFor <http://www.Department0.University0.edu> .
}
```

图5　StarQuery1的具体查询内容

Fig. 5　The specific query content of StarQuery1

从图6中可以看出,随着数据集的不断增大,由于两阶段索引的查询范围只与数据的邻域信息的大小有关,因此两阶段索引的查询范围受数据规模的影响相对较小,具有较好的稳定性.

在上述查询的基础上,图7进一步增加了时间信息,即时间查询变量"? t_t",且该查询StarQuery2需要进行更复杂的连接操作.图8给出了相应的实验结果.

可以看出,两阶段索引通过获得候选集并对候选集进行验证的方式避免了复杂的连接操作,查询性能并没有明显下降.同时,在两阶段索引的第一阶段索引中是通过提取查询条件中的邻域信息来进行候选集的筛选.如果给出的查询条件更多,能够提取的信息也更多,更有利于在第一阶段索引时缩小候选集的范围.因此,两阶段索引在处理复杂的星型查询上更有优势.

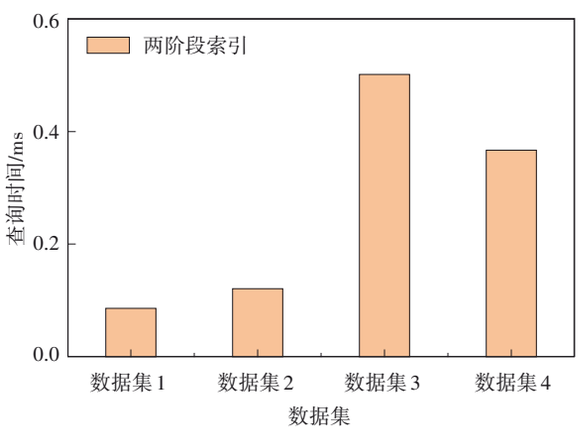

图6　StarQuery1的实验结果

Fig. 6　The results of StarQuery1

```
Prefix org=<http://example.org/uni#>
Prefix dep=<http://www.Department0.University0.edu/>
Prefix org=<http://example.org/uni#>
Prefix rdf=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?x, ?y1, ?t_t.
where {
  ?x rdf:type org:FullProfessor ?t_t.
  ?x org:worksFor <http://www.Department0.University0.edu> .
  ?x org:teachOf dep:Course0.
  ?x org:name ?y1 .
}
```

图7　StarQuery2的具体查询内容

Fig. 7　The specific query content of StarQuery2

图9中的StarQuery3进一步添加了有效时间t_v和事务时间t_t的双时态约束.图10展示了在两阶段索引和KD树索引下StarQuery3在各数据集上的表现.

从图10可以看出,在时间信息已知时,两阶段索引除了需要对三元组数据进行验证,同时还需要对双时态信息进行验证,因此图10中两阶段索引在StarQuery3上所花费的时间略多于图6中的StarQuery1.此外,基于KD树的索引通过查询条件的时间信息和谓语宾语信息可以获得1个候选集,但因为另1个查询条件中并没有出现时间

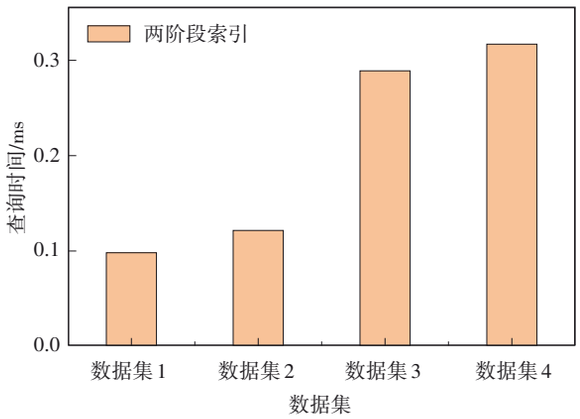

图8 StarQuery2的实验结果

Fig. 8 The results of StarQuery2

```
Prefix org=<http://example.org/uni#>
Prefix rdf=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select ?x
where {
  ?x rdf:type org:AssociateProfessor 2014.3<vt<2018.12 tt>2020.9.
  ?x org:worksFor <http://www.Department0.University0.edu> .
}
```

图9 StarQuery3的具体查询内容

Fig. 9 The specific query content of StarQuery3

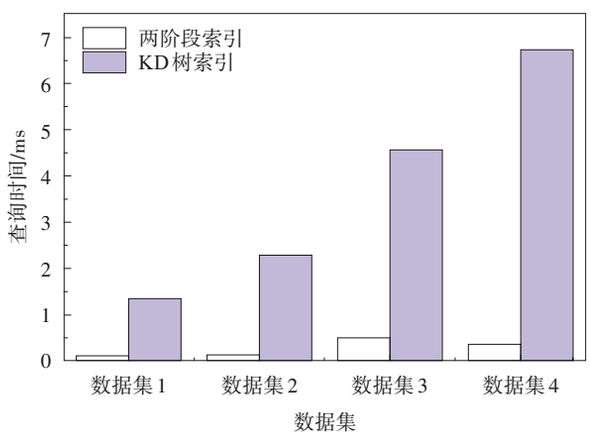

图10 StarQuery3的实验结果

Fig. 10 The results of StarQuery3

信息,此时就需要在KD树中分别找到候选集中的数据所在的位置来进行进一步的验证,因此基于KD树的索引所花费的时间较长,而本文提出的两阶段双时态索引查询效率更高.

从上述实验结果可以看出,本文提出的针对双时态RDF星型查询的两阶段索引在处理星型查询时更有优势,同时随着数据集规模的增大,索引的查询效率具有较好的稳定性.

4.3 索引构建时间和占用空间性能实验分析

为了进一步验证索引的性能,本文从索引的构建时间和占用空间两方面对提出的两阶段索引进行了评估.首先,在构建时间方面,图11展示了基于KD树的索引和本文提出的两阶段索引的构建时间.从图中可以看出,由于两阶段索引在索引构建前需要先对邻域信息进行提取,同时由于基于KD树的索引是针对单时态的索引,因此本文提出的两阶段索引在整体上花费的时间略多.

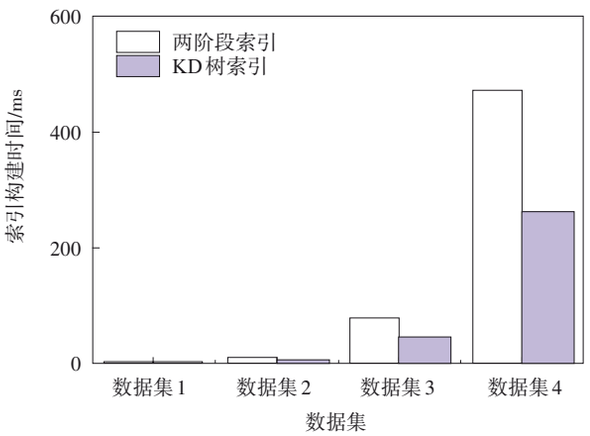

图11 索引构建时间

Fig. 11 The building time of indexes

图12展示了索引的占用空间情况.由图12可知,基于KD树的索引在构建位图时,分别构建了3个不同的位图索引,因此数据也需要多次存储.而在两阶段索引中,并不存在同一数据多次存储的情况.因此,两阶段索引更节省存储空间.

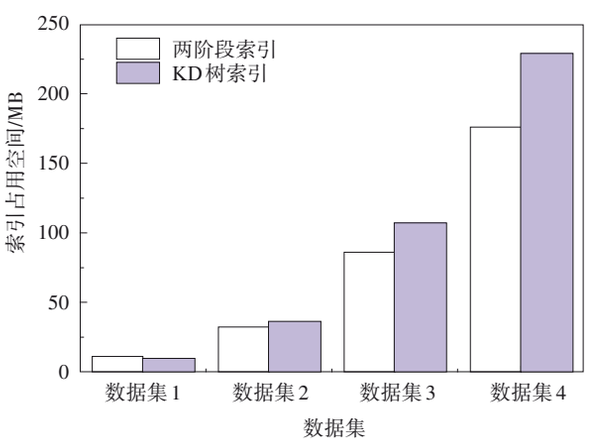

图12 索引占用的空间

Fig. 12 The storage space of indexes

5 结 语

本文针对双时态RDF的管理需求,提出了1种针对双时态RDF星型查询的两阶段索引方法.第一阶段索引提出1种基于双时态RDF节点邻域信息的候选集生成方法,第二阶段索引进一步

提出1种基于位图索引的结果集筛选和生成方法.进一步地,本文在查询效率和索引性能方面进行了对比实验分析,实验结果验证了本文所提索引的有效性.今后的研究工作将对双时态RDF索引技术作进一步的扩展研究,并考虑进一步引入缓存和空间压缩技术来解决索引的性能问题.

参考文献:

[1] Manola F, Miller E. RDF 1.1 primer [EB/OL]. (2014-06-24) [2023-03-15]. https://www.w3.org/ TR/rdf11-primer.

[2] Gutierrez C, Hurtado C, Vaisman A. Temporal RDF [M]// Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2005: 93-107.

[3] Pugliese A, Udrea O, Subrahmanian V S. Scaling RDF with time [C]// International Conference on World Wide Web. New York: Association for Computing Machinery, 2008: 605-614.

[4] Wang H T, Tansel A U. Temporal extensions to RDF [J]. *Journal of Web Engineering*, 2019, 18(3): 125-168.

[5] Runge L, May W. Towards semantic identification of temporal data in RDF [C]// Proceedings of the 34th Workshop on Basics of Database Systems. Calw: CEUR, 2023: 1-8.

[6] Yan L, Zhang Z Q, Yang D. Temporal RDF (S) data storage and query with HBase [J]. *Journal of Computing and Information Technology*, 2020, 27(4): 17-30.

[7] Bellamy-McIntyre J. LSPARQL: transaction time queries in RDF [D]. Auckland: The University of Auckland, 2020.

[8] Li H X. A new query method for the temporal RDF model RDFMT based on SPARQL [C]// Artificial Intelligence and Information Systems. New York: Association for Computing Machinery, 2021: 1-6.

[9] Meijer L. Bi-VAKs: bi-temporal versioning approach for knowledge graphs [D]. Delft: Delft University of Technology, 2022.

[10] Ma R Z, Han X, Yan L, et al. Modeling and querying temporal RDF knowledge graphs with relational databases [J]. *Journal of Intelligent Information Systems*, 2023, 61(2): 569-609.

[11] Chawla T, Singh G, Pilli E S, et al. Storage, partitioning, indexing and retrieval in big RDF frameworks: a survey [J]. *Computer Science Review*, 2020, 38: 100309.

[12] Zhao P, Yan L. A methodology for indexing temporal RDF data [J]. *Journal of Information Science & Engineering*, 2019, 35(4): 923-934.

[13] Zhou T Y, Liu Y, Zhang H, et al. Optimization of bit matrix index for temporal RDF [M]//Communications in Computer and Information Science. Singapore: Springer Nature Singapore, 2022: 95-107.

[14] Zang S Q, Han S, Yuan P P, et al. HyperBit: a temporal graph store for fast answering queries [J]. *Data & Knowledge Engineering*, 2023, 144: 102128.

[15] Zhang F, Li Z Y, Peng D H, et al. RDF for temporal data management: a survey [J]. *Earth Science Informatics*, 2021, 14(2): 563-599.

[16] Chawla T, Singh G, Pilli E S, et al. Research issues in RDF management systems [C]// Emerging Trends in Communication Technologies. Piscataway: IEEE, 2016: 1-5.

[17] Guo Y B, Pan Z X, Heflin J. LUBM: a benchmark for OWL knowledge base systems [J]. *Journal of Web Semantics*, 2005, 3(2/3): 158-182.