计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
49(10): 2131－2141, 2012

高可扩展的 RDF 数据存储系统

袁平鹏　刘　谱　张文娅　吴步文
(服务计算技术与系统教育部重点实验室(华中科技大学)　武汉　430074)
(集群与网格计算湖北省重点实验室(华中科技大学)　武汉　430074)
(华中科技大学计算机科学与技术学院　武汉　430074)
(ppyuan@hust.edu.cn)

A Highly Scalable RDF Data Storage System

Yuan Pingpeng, Liu Pu, Zhang Wenya, and Wu Buwen
(*Key Laboratory of Services Computing Technology and System (Huazhong University of Science and Technology), Ministry of Education, Wuhan* 430074)
(*Key Laboratory of Cluster and Grid Computing of Hubei Province (Huazhong University of Science and Technology), Wuhan* 430074)
(*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan* 430074)

Abstract　As RDF(Resource Description Framework) is flexible to express and easy to interchange, the volume of RDF data is increasing rapidly. TripleBit aims to propose an efficient approach in data storage and query processing for large scale RDF data in several aspects. TripleBit employs delta compression and variable integer encoding schemes in order to reduce the storage space. The data tables are partitioned into several chunks, which not only facilitate the buffer management but also make the data more compact, therefore it can accelerate the query processing. We employ heuristic rules to generate query plan dynamically. Besides, two-stage execution strategy is used in multiple-variable query which can reduce the intermediate result. The performance evaluation is compared with the state of art RDF stores, such as RDF-3X, MonetDB. Experimental results demonstrate that TripleBit saves at least 40% storage space while the speed of query processing has been improved very much.

Key words　RDF; semantic data storage; data encoding; query processing; query plan

摘　要　由于资源描述框架(resource description framework, RDF)具有表达灵活、简洁等优点,已被接受为表达元数据及万维网上数据互联的规范.近年来,其数据量在以飞快的速度增长.相应地,要求存储RDF数据的系统应具有高扩展性.介绍了一个高可扩展的RDF数据存储系统TripleBit.为尽可能降低存储空间消耗,采用了增量压缩和变长整数编码方法.并采用了数据分块的存储方法,既使得存储管理方便又使得存储结构紧凑,加速了数据读取.系统提供了基于启发式规则的动态查询计划生成方法,所产生的查询计划在执行过程中根据中间结果会相应作调整,以保持最优的执行顺序.对于多变量的查询,使用二步执行策略以减少查询过程中产生的中间结果.与目前流行RDF数据存储系统相比较,在存储空间上RDF-3X比TripleBit至少多40%;在查询性能上,比RDF-3X和MonetDB获得数倍的提升.

关键词　资源描述框架;语义数据存储;数据编码;查询处理;查询计划

中图法分类号　TP392

收稿日期:2012-06-05;**修回日期**:2012-07-23
基金项目:国家自然科学基金项目(61073096);国家"八六三"高技术研究发展计划基金项目(2012AA011003)

资源描述框架(resource description framework, RDF)[1]为描述万维网上的各种资源提供了一个统一的规范. 根据 RDF,一个陈述(事实)通常包含主体(subject)、谓词/属性(predicate/property)和客体/属性值(object/property value),即所谓三元组. 由于 RDF 的灵活性和简洁性,在许多领域 RDF 成为表达元数据的基本方式,如生命科学、化学、商业智能(business intelligence)和社交网络等,越来越多的机构采用它来表达它们的元数据. 纽约时报、美国白宫等应用 RDFa、微数据(microdata)等技术来表达它们网页的元数据,帮助互联网应用获取网页内容的元数据. 搜索引擎公司也极力提倡在网页中应用语义网技术,如 Google 能够识别网页里的元数据. 不仅如此,微软公司旗下的搜索引擎 Bing 和 Google 先后收购了语义网数据库技术公司,利用它们所发展的技术帮助搜索引擎增强搜索的精确程度. 这些都促进了 RDF 的广泛应用. 随着 RDF 被广泛接受为一种数据表达规范,RDF 数据量飞速增长. 据统计,到 2011 年 9 月止,仅在 W3C 的开放链接数据项目(linked open data)中发布的 RDF 数据就达到了 310 亿条[2]. 该数据量是 2010 年 1 月份的 130 亿条的 2 倍多,是 2008 年 4 月份数据量的 15 倍.

如何高效地管理这些 RDF 数据成为亟待解决的问题. 目前一种比较流行的存储方式是将关系数据库作为存储后端. 由于 RDF 数据比较稀疏(数据规模大时尤其明显),用关系数据库存储和管理性能较差. 更多地研究开发了适应 RDF 数据特性的原生存储系统(Native store).

目前学术界已开发了一些原生 RDF 存储系统[3],如 Sesame[4], HexaStore[5], RDF-3X[6-7]等. 但是只有少数几个系统如 RDF-3X 具有较好的扩展性[6-8],但 RDF-3X 的扩展性是以牺牲较多的存储空间来换取. 当数据量增长时它的存储开销增长迅速. 并且较多的冗余存储导致查询执行器需要选择从哪份存储中读取数据,因此导致在生成查询计划和执行过程中消耗大量的时间.

针对目前 RDF 存储系统研究的现状,本文研究了 TripleBit 系统. 该系统旨在应对 RDF 数据的数量增长,开发具有高扩展性能的大规模 RDF 数据存储系统. 系统能够应对 Billion 级别的 RDF 数据存储需求,并且在查询性能损失不大的前提下,运用适当的压缩技术降低数据所占用的空间,从而获得更高的性能提升. 此外,系统能够快速地处理用户所提交的查询.

1 研究现状

目前的 RDF 数据存储系统按存储方式大体上可以将其分成 4 类:三元组存储、属性表、列存储与垂直划分相结合方式和图存储方式.

1) 三元组存储

RDF 数据是由 3 个分量(主体、谓词和客体)构成的,在存储时直接将其 3 个分量存储为一个含有 3 个数据项的大表,称之为三元组表. 鉴于该方式将一条 RDF 数据在表中存储为一行,所以通常又被称为行表. 三元组表存储方式非常简单. 但由于只有单一表存放数据,因此会产生很多的表自身的连接操作(self-join). 一般来说,RDF 数据中谓词的数量通常很少,例如 LUBM[9-10]数据集中谓词只有 18 个. 因此,三元组表存储方式会将谓词存储很多次,产生巨大的冗余. 并且,当数据量增大时,三元组表会变得非常庞大,这将极大降低表扫描性能. 针对上述问题,采用比较多的方法是重复存储数据和建立辅助的索引来加速数据的访问. 如 RDF-3X[6-7]和 HexaStore[5]将 RDF 数据的可能组合方式都存储起来. 不仅如此,RDF-3X 还存储了 SP, SO, PS, PO, OS, OP, S, O, P 等 9 种统计信息,以辅助生成较优的查询计划. 存储多份数据必然导致对存储空间需求的增长. 为此,RDF-3X 采用 Delta 压缩的方式来减少空间的占用量.

2) 属性表

Wilkinson 等人[11-12]提出了将 RDF 数据用属性表的方式进行存储. 属性表第 1 列对应的是 RDF 数据中的主体,其余列对应主体所具有的属性. 每一行中依次对应主体及其相应属性值. 相对于三元组表瘦长,属性表要肥大些. 若采用单个属性表来存储语义数据,由于 RDF 数据比较稀疏,表中会有很多空值,造成存储空间浪费. 为了紧凑存储,可以根据属性将实体分成多个属性表存储. 具体的划分方法有两类:聚集属性划分和实体类型划分. 聚集属性划分的方式按照属性之间的关联关系将属性划分成若干组,然后将实体按照相应的属性分别存储到这些表中. Oracle[13]也使用基于属性表的方式来加速 RDF 数据的查询,但与 Jena2 不同的是它使用视图的方式来实现.

虽然采用属性表减少了连接操作,但是其缺点也不容忽视:首先,在 RDF 数据中多值属性是很常见的情况. 无论属性如何划分,在存储多值的属性表

中还是会有大量的空值,这对存储和查询都会有很大的影响[13-15];其次,当查询中含有谓词未知的查询模式时需要扫描所有的属性表,这会带来巨大的开销.实验表明[16],当属性表的扩展性不好及数据量增大时会导致存储效率和查询性能急剧下降.

3) 列存储与垂直划分相结合的存储方式

由于在基于属性表存储方式中实体的公共属性很难确定,导致数据的划分没有一个固定的标准,使得用户在此类 RDF 数据存储系统中存储数据时不方便. Abadi 等人[8,17]提出了根据属性将 RDF 数据进行垂直划分,然后存于列式数据库中.因此,该方式称之为列存储与垂直划分相结合的存储方式.在该存储方式中,三元组被存储在一组二维表中,二维表的个数等于谓词的个数.在这些表中,第 1 列存储的是拥有相应谓词的主体,第 2 列存储的是该主体所对应的客体.

使用该方式来存储三元组具有多个优势[8],如可以高效地存储多值属性,并且可以减少 I/O 的次数,这是因为在选取某个属性值时只需读取相应的数据表,不必访问无关的属性值.

但是该方式也有一定的缺陷:首先,当 RDF 数据集中的谓词数增大时,实验[18-19]表明其查询性能比使用基于三元组表的方式差;其次,当查询中含有未知的谓词时,查询执行将要扫描所有数据表.因此,使用垂直分割的方式主要需要解决谓词的索引问题来加速有关的查询.

4) 图存储方式

Bonstrom 等人[20]将 RDF 数据作为图存储在对象数据库中;Kim 等人[21]和 Matono 等人[22]提出了基于路径的方式存储 RDF 数据;Wu[23]将超图的数据模型作为 RDF 数据的持久化存储策略;Yan 等人[24]提出将 RDF 图分割成若干个子图并辅以 Bloom Filter 之类的索引,在查询时能够快速地确定要查的数据是否在相应的子图中.但是这些方法主要解决的问题是路径查询没有考虑扩展性和高效性问题[5]. gStore[25]采用邻接表的方式将属于同一个主体的多个属性值链接在一起,这样在查询一个主体的各个属性值时可以快速得到,并且能够适应谓词变化的情况.

BitMat[26]将 RDF 数据表达成三维比特矩阵,每一维分别对应 S,P,O.矩阵中的元素为比特"0"或者"1".具体取值取决于 S,P,O 中的实例 s,p,o 之间是否存在关系.然后将此三维矩阵延特定维(如 P 维)切分成若干二维矩阵.对二维矩阵按行采用 D-gap 进行压缩存储.由于采用的是 D-gap 的压缩方式,扫描数据及查询的效率都不是很高.

查询计划生成和查询计划执行对查询处理的高效执行有着重要的作用.查询计划生成采用的一种方式是使用动态规划(dynamic programming)找到最优的查询计划[6-7].但是,采用动态规划的方法是一个耗时的过程.在查询执行的过程中,RDF-3X 使用旁路信息传递(sideways information passing)的方式来减少中间结果[6-7]. Vidal 等人提出了按照查询变量将查询语句中的查询模式划分成多个星型模式组,然后进行优化[27].吕彬等人[28]提出利用本体信息自动计算属性相关性的方法,从而调整连接操作的选择度估计值.

2 TripleBit 设计原则

围绕大规模 RDF 数据存储及处理问题,本文在设计 TripleBit 时着眼于如下几个方面来提升 RDF 存储系统的性能.

1) 减少存储空间消耗.首先,虽然存储价格很便宜,现代计算机普遍配备有大容量内存,但现代计算机存取数据所需要的时间大于加法和乘法运算,因此一次存取尽可能多的数据将会提高性能;其次,计算机的存储容量是有限的.过高的存储需求必然导致内存与磁盘间交换频繁,使得性能降低;第三、存储空间降低可以提高在读取数据过程中 CPU 缓存的命中率,从而提高系统的性能.基于以上分析,设计 TripleBit 时减少不必要数据的存储.现有的 RDF 存储系统,如 RDF-3X,HexaStore 一般存储主体、谓词和客体所有可能排列(SPO,SOP,PSO,POS,OSP,OPS)来加速查询.同这些系统相比,TripleBit 只存储 PSO 和 POS 两种组合.相比于 RDF-3X 等存储 6 种组合,减少了大量存储空间的消耗.此外,通过压缩或编码技术来降低存储空间需求.针对不同的数据结构和数据的用途采用不同的压缩算法.对于经常使用的数据,在压缩数据的同时还要考虑到数据在解压时和数据定位的开销,对这种类型的数据采用轻量级的压缩策略.

2) 数据分块存储加速数据的读取速度.数据分块存储可以实现紧凑存储,按照磁盘块大小和内存页大小对将要存储的数据分割成多个数据块.采用磁盘块大小分片的好处是可以充分利用文件系统中的缓冲功能,在加载数据时进一步减少磁盘 I/O 次数.采用内存页大小进行分片的好处是,在读取数据

时可以减少系统在地址翻译时的页表查询次数,即提高 TLB 的命中率. 在实际设计时,数据块的大小为磁盘块大小,这是因为磁盘块的大小通常是内存页大小的整数倍.

3) 加速查询处理速度. 首先,为了加速查询处理速度,TripleBit 设计了高效的查询算子,对扫描数据操作而言,针对不同查询模式要求设计相应的扫描数据方式;执行连接查询时,根据参与连接数据表特征的不同,采用不同的连接操作方式;其次,如何估计每个查询模式和连接运算的计算代价及其计算效率对查询计划的生成有重要的影响,在 TripleBit 中使用结合统计信息和连接的类型的方法来进行估算;最后,在生成查询计划时,在较短的时间内按照一定的启发式规则生成查询计划,并在查询过程中,根据实际情况不断地对计划进行修改. 在执行查询时先将候选集中无用的数据集去掉,减少中间结果,在此基础上得出最终的结果.

3 RDF 数据存储

RDF 数据由若干条语句构成. 一条语句是由资源、属性类型、属性值构成的三元组,表示资源具有的一个属性. RDF 原始数据为文本数据. 每个实体通常有多个属性,即在原始的数据中表示一个实体的 URI 出现多次. 若将 URI 作为实体的 ID 并将其直接存储会极大浪费空间,而且对于实体匹配极其不方便. 因此,在存储时一般先将 URI 或者字符串转换成整形标识符(ID). 同时,URI 可能有相同子字符串,对此将 URI 进行压缩存储. 这样每一条 RDF 数据转化成$\langle sid, pid, oid\rangle$3 个整数构成的元组. 考虑到谓词或属性的数目相对于主体和客体来说很少,TripleBit 将谓词相同的元组存放在一起. 因此实际存储时只需要存储$\langle sid, oid\rangle$对. 下面将介绍 URI 的压缩存储和转换之后的二元组对的存储.

3.1 URI 压缩

URI(universal resource identifier)用来唯一地标识 RDF 数据中的实体. 在一个 RDF 数据集中,多个实体的 URI 可能有公共前缀. 区分公共前缀的一个依据就是以某些字符为标志的分隔符,比如最常见的以"/"为分隔符,形成了 URI 的层级结构;还有一类是以"#"为分隔符,这类分隔符常出现在表示实体类型的三元组中. TripleBit 中采用的 URI 压缩主要是基于这两类分隔符进行压缩. TripleBit 利用 URI 中最后一个字符"/"或者"#"为分割点,将 URI 分成前缀和后缀. 并建立 URI 前缀和后缀和相应 ID 间的映射表.

采用该压缩方案可以很快地查找到一个 URI 的 ID. 具体步骤为:首先,将 URI 分割成前缀和后缀;然后,将前缀在前缀字符串 ID 映射表中进行查找,如果没有查找到则返回,否则将前缀 ID 和后缀字符串结合起来在后缀字符串 ID 映射表中进行查找,并返回结果. 根据 ID 查找 URI 的步骤为:首先,根据 ID 在后缀字符串 ID 映射表中查找出前缀 ID 和后缀字符串;然后,根据前缀 ID 在前缀字符串 ID 映射表中查找到前缀;最后将前缀和后缀合并即可得到最终的 URI.

3.2 RDF 数据编码存储

如上所述,对于每个三元组可采用$\langle sid, oid\rangle$对来记录. 存储时按先 sid 后 oid 进行排序存储. 此种存储组织适合于以主体为关键字的查询,若运行以客体作为关键字的查询则性能差. 为此,TripleBit 同时存储另一种组合,即$\langle oid, sid\rangle$. 为了叙述方便,这里不区分主体 ID 和客体 ID,统一将值对称为$\langle x, y\rangle$对. 实际存储时采用增量压缩. 即 x 表示值对中较小的 ID,y 为较大 ID 与较小 ID 之间的差值. 由于查询时一般均返回值对,所以此压缩所带来的开销是极其微小的. 值对中较小 ID 并不一定是主体或者客体. 为了区分,根据 sid 是否小于 oid,将值对归类存放在一起. 每种组合存储在一桶(bucket)中. 每一桶进一步划分为块(chunk).

通常在计算机中一个整数所占用的空间为 32 b. 对于小整数高位字节通常为 0. 不是所有整数在实际存储时都会将 4 B 全部用到. 因此,TripleBit 存储 ID 时采用变长编码的方式. 具体地,在编码时,使用最高位作为标志位. 其作用有两个方面:一方面表示该 ID 是否结束,另一方面表示该 ID 是主体还是客体. 在具体存储时,采用"0"表示主体,"1"表示客体. 例如,当实体的 $ID=10$ 时,如果采用该压缩方式,则应该表示为"00001010"(当该 ID 在三元组中是主体时)或者"10001010"(当该 ID 在三元组中是客体时). 类似地,当 $ID=320$ 时,会被存储为"00000010 01000000"(主体)或者"10000010 11000000"(客体). ID 中每个字节的最高比特位具体采用"0"或"1"取决于该实体在相应三元组的角色. 如果 $ID<2^7$,采用这种压缩方式每个 ID 可以节省 3 B;如果 ID 在 $2^7\sim2^{14}$之间时,则每个 ID 可以节省 2 B. 可以将频繁实体分配较小的 ID,非频繁实体采用较大 ID,这样可以进一步减少存储的空间.

由于 x 和 y 是相邻存储,所以只要两个相邻字节的最高位的值不同 ID 的类型就发生变化.例如,设三元组 SO 对为〈1,5〉,则可以计算出采用该压缩模式下的 $x-y$ 对的值为〈1,4〉.进一步,采用变长整数编码后,该三元组表示为"00000001 10000100".

采用这种压缩编码方式,每个三元组只需要 2～8 B 来进行存储(一般来说,RDF 数据集中的三元组的数目比实体的个数要大得多).表 2 显示 TripleBit 存储每个三元组只需 5.02～5.97 B.与直接存储 3 个 ID 的方式相比,TripleBit 节省了一半存储空间.此外,采用这种压缩方式还有如下优点:由于在压缩时,单个三元组数据与其他的三元组无关,因此能够快速解压一个三元组数据;在解压得到具体的数据时只经过缓存一次,大大提高了缓存的命中率.

在一个数据块中,查找特定的数据时可以直接采用二分查找,无需提前解压.在压缩的数据块中搜索特定的 ID 时,由于 y 值为两个 ID 之间的差值,所以要么是查找等于 x 的 ID 值,要么是查找等于 $x+y$ 的 ID 值.由于〈x,y〉对是有序排列存储,块中搜索采用二分查找.具体地,当指针所指向的数据不是〈x,y〉对的开头时,需要判断每个字节的最高位的位值,回溯到最高位的位值为"1"的字节时停止.然后通过最高位来判断〈x,y〉对值的结束位置.通过不断的移位来得到〈x,y〉对值.因此,在块内搜索一个特定的 ID 值时其时间复杂度为 $O(\log n)$.

4 查询处理

4.1 查询表示

在 TripleBit 中,使用查询图表示查询.在查询图中,如果两个或者两个以上的查询模式中含有同一个变量,则将该变量称为公共变量,并称这两个查询模式之间具有连接关系.如果在一个查询模式中出现两个或者两个以上的公共变量,则称这两个公共变量有依赖关系.通过扫描每个查询模式的各个分量,可以得出与一个特定的变量相联系的查询模式.从而得出公共变量的个数以及公共变量之间的依赖关系,进而可以得出查询的类型.在多个查询模式组成的查询中,根据公共变量之间的关系将查询大致分成 3 类:星型查询、多变量非循环查询和多变量循环查询.

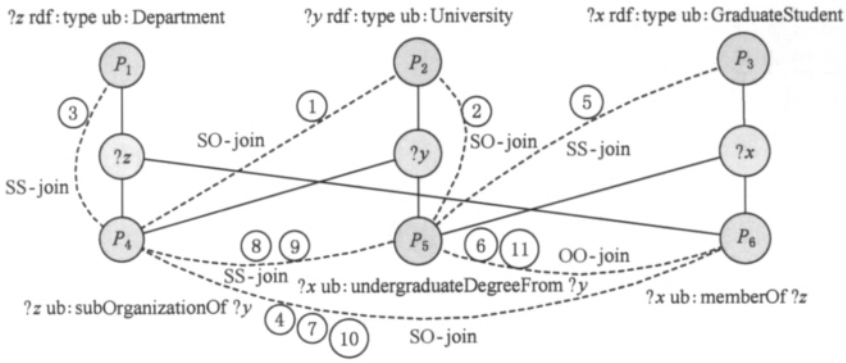

Fig. 1 Join ordering.

图 1 查询连接执行顺序

例如附录中查询 Q_5 可以表达成图 1 的形式.在图中用直线表示公共变量与查询模式之间的关系.从图 1 可以看出,变量 ?x 与查询模式 P_3,P_5 和 P_6 有联系.即变量 ?x 在此 3 个查询模式中出现.变量 ?x 与变量 ?y 之间、变量 ?x 与变量 ?z 之间和变量 ?y 与变量 ?z 之间都有依赖关系,即 3 个公共变量之间的依赖关系是循环的,因此,该查询是一个多变量循环查询.

4.2 选择性估计

选择性(selectivity)估计包括单一查询模式的选择性估计和查询连接模式之间连接的选择性估计.共有 7 种单一查询模式:(?s p ?o),(s p ?o),(?s p o),(s ?p o),(s ?p ?o),(?s ?p o).单一查询模式(s ?p ?o),(s p ?o),(?s ?p o),(?s p o)的选择性估计可通过查询统计信息表来获得.系统预先存储了 4 个统计信息表,称为 S(subject)表、O(object)表、SP(subject-predicate)表、OP(object-predicate)表.S 表和 O 表分别记录每个主体所对应的三元组个数、每个客体所对应的三元组个数.相应地,SP 表、OP 表记录特定主体-谓词对所对应的三元组个数、特定客体-谓词对所对应的三元组个数.因此,以上 4 个查询模式对应的三元组数目已经保存在统计表中.因此,4 种模式的选择性可直接查询统计信息表得到.

剩余的 3 个单一查询模式实际上也容易得到. 对于形如(?s ?p ?o)的查询模式,由于其候选集是所有的三元组,所以选择性值为系统中保存的所有三元组的个数. 对于形如(?s p ?o)的查询模式,由于是按照 P 划分,同样可得到该模式所对应三元组的数目. 由于在实际 RDF 数据中两个实体之间关系数量有限,所以可以将形如(s ?p o)的查询模式的选择性定为一个常数. 对于形如(s p o)的查询模式,由于 3 个分量都是已知的,所以直接将其选择性定为 1.

查询连接模式是由两个单一模式构成的连接查询. 连接查询的选择性根据参与连接的两个单一查询模式的选择性来进行估计. 连接查询有 6 种:主体-主体连接;主体-客体连接;主体-谓词连接;谓词-谓词连接;谓词-客体连接;客体-客体连接,其中主体/客体同时是谓词的情况比较少见. 因此主体-谓词连接、客体-谓词连接的选择性一般较好. 另外,谓词的数目一般小于主体或者客体的数目,因此谓词-谓词连接的选择性也优于主体与客体之间的连接. 但这种定性选择性评估并不是绝对的,当参与连接的两个模式有大量候选结果时原来选择性优变差. 例如连接查询(?s p o_1. ?s_1 ?s ?o)两个子查询模式所对应的中间结果集都很大时,连接查询的选择性就比较差. 因此,这里采用参与连接两种模式的选择性值之积来作为连接的选择性值.

4.3 动态查询计划生成

生成查询计划的过程就是对查询进行优化的过程. 最优的查询计划是按照该计划去执行查询所耗时间最短的计划. 但是,这需要不断地对查询模式的各种组合进行枚举,对每种组合计算其运行时的代价,直至得到最优的结果. 显然,精确地计算代价开销太大,在实际中采用了基于代价估计的方式. 一种方法是采用动态规划的方式来找到估计代价最低的执行计划,但是其时间复杂度太高. 因此,这里采用选择性估计来动态生成查询计划.

在估算出各个查询模式和查询连接的选择性之后,需要在此基础上调节查询模式的执行顺序,即生成查询计划. 在生成查询计划时,基于公共变量对查询模式的执行分组优化. 具体的算法如图 2 所示.

在调整查询模式的执行顺序时,首先,查询每个查询模式的选择性值(算法 1 行②). 然后,生成每个公共变量的选择性:使用每个公共变量所对应的查询模式的选择性的最小值作为该公共变量的选择性. 然后,将公共变量按照选择性的大小进行排序,以确定每个变量的执行先后顺序. 最后对每个公共变量执行如下操作:生成该公共变量对应的具有最小选择性的查询模式与其他查询模式连接操作的选择性;根据这些选择性对公共变量所对应的查询模式进行排序;生成各个查询模式之间的连接操作类型.

```
Algorithm 1. Query plan generation.
Input: queryStr[in], queryGraph[in, out];
Output: queryGraph-an optimized query graph.
① for i←0 to queryGraph. pattern. size-1
②   do getPatternSelectivity(queryGraph. pattern[i]);
③     for i←0 to queryGraph. joinVariable. size-1
④     do getVariableSelectivity
          (queryGraph. joinVariable[i]);
⑤       sortVariableBySelectivity
          (queryGraph. joinVariable);
⑥     for each var in queryGraph. joinVariable
⑦       do generateJoinSelectivity(var);
⑧         sortPatternBySelectivity(var);
⑨         generateJoinType(var)
```

Fig. 2 Query plan generation.

图 2 查询计划生成

在生成的优化查询计划图中,查询执行顺序分成两种:变量的执行顺序和查询模式的执行顺序. 变量的执行顺序优先于查询模式的执行顺序. 这样就可以将一个变量对应的查询模式一起执行完. 在关联矩阵的数据块中读取出来的数据是关于键值有序的,这样就可以利用归并连接算子来降低连接运算的开销. 并且当具有最小的选择性的变量所对应的查询模式优先执行时,可以降低在查询过程中的中间结果集大小.

上述过程是初步查询计划生成过程. 在 TripleBit 中,查询计划在执行过程中根据中间结果的信息会作动态调整.

4.4 查询计划执行

在 TripleBit 中查询计划的执行是一个动态过程,即在具体的执行过程中动态调整执行计划. 具体的做法是根据上一轮连接运算执行的结果动态地对下一轮连接执行顺序进行调整. 如例子查询的最终连接运算执行顺序如图 1 所示.

该查询的执行一共进行了 3 轮 join. 除了第 1 轮的查询计划是初始生成以外,其余都是动态生成. 图 1 中行①~⑥的连接为第 1 轮,该顺序是查询计划根据统计信息估计选择性而生成的. 第 2 轮连接操作行⑦~⑧,其顺序是根据第 1 轮的执行结果而生成的. 由于查询模式 P_4 所对应的中间结果是最小的,为了尽量削减中间结果,将其他查询模式的中间结果都与其进行连接操作. 第 3 轮连接操作行⑨~

⑪,其顺序是根据第 2 轮连接执行的结果而生成的,采取该执行顺序的原因同第 2 轮.

5 实验结果与分析

5.1 数据集和对比系统

本文所采用的对比系统是 RDF-3X 和 MonetDB.这两个系统分别是两种典型的 RDF 存储方式:Native store 和列式关系数据库系统[29].目前的公开文献显示 RDF-3X 是性能最好的系统之一.MonetDB 是成熟的列式关系数据库,它是 C-Store[29]的延续.Abadi 等人推荐采用 MonetDB 作为 C-Store 的替代系统.因此,以上两个系统被广泛地用在 RDF 存储系统性能对比测试中[8,25-26].RDF-3X 的版本是 0.3.5,MonetDB 版本是 2011 Aug. SP1.3 个系统都使用 gcc 的-O2 选项进行优化.

各系统均使用 GCC 4.4.4 进行编译.为了评估 3 个系统的查询性能,采用了 6 个 LUBM 查询(见附录).对每个查询,每个参与测试的系统都分别在 Cold Cache 和 Warm Cache 两种条件下运行.对于 Warm Cache,为了减少实验的误差,每个查询都运行 5 次,并取算术平均值作为最终的结果.本文使用 LUBM[9-10]提供的数据生成器产生了 10 M(包含 13 879 970 条三元组),50 M(包含 69 099 760 条三元组),100 M(包含 138 318 414 条三元组),500 M(包含 691 085 836 条三元组),1 B(包含 1 335 081 176 条三元组)规模级别的数据集.

实验主要运行在两台服务器上.数据集 LUMB-10 M,LUMB-50 M,LUMB-100 M,LUMB-500 M 所运行的服务器硬件配置为 16 核 Intel Xeon E5620 型 CPU,其主频为 2.40 GHz;内存大小为 24 GB;磁盘空间为 1 TB.机器所使用的操作系统为 CentOS 6.0,内核版本号为 2.6.32.由于 MonetDB 当装载大数据集时需要内存更大的机器,因此数据集 LUMB-1 B 所运行的服务器配置为 4 路 4 核服务器、64 GB 内存、两块 15 000 转的 300 GB 的 SAS 磁盘.CPU 是 Intel Xeon CPU E7420(2.13 GHz).所采用的操作系统为 Red Hat Enterprise Linux Server 5.1(内核版本 2.6.18).磁盘交换空间为 64 GB.该服务器与一由 20 块 1TB 的 7 200 转磁盘组成的磁盘阵列相连.

5.2 存储空间比较

下面主要比较 3 个系统的静态存储空间,即对比 RDF-3X,MonetDB,TripleBit 3 个系统的数据文件大小(如表 1 所示).

Table 1 Comparing Storage of Three System

表 1 存储空间比较 GB

Data Set	RDF-3X	MonetDB	TripleBit
LUMB-10 M	0.67	0.35	0.42
LUMB-50 M	3.35	1.7	2.39
LUMB-100 M	6.83	3.50	4.88
LUMB-500 M	34.84	22.8	22.01
LUMB-1 B	69.89	45.6	44.5

根据表 1 结果,TripleBit 所占有的存储空间要小于 RDF-3X 所占有的存储空间.TripleBit 的存储空间需求比 RDF-3X 的存储空间需求降低了 29%~37%.这是因为 RDF-3X 将三元组可能的 6 种组合都进行了存储,并且为了得到统计信息,还存储了 9 个辅助的统计信息表.但是在 TripleBit 中,只是存储了两份原始数据和 4 个统计信息表,并通过压缩方式来减少存储空间.TripleBit 和 MonetDB 占用的存储空间大致相同.这是因为 MonetDB 只存一份数据.

对比 LUBM-10 M~LUBM-1 B 等 5 个数据集存储空间大小可知,TripleBit 在存储空间消耗上有着良好的扩展性.随着数据的增长,占用的存储空间基本上呈线性增长.虽然 RDF-3X 存储增长也呈线性,但由于单位存储消耗的空间比 TripleBit 多,所以存储增长的幅度大于 TripleBit 的增长幅度.相反,MonetDB 的存储消耗随着数据规模加大呈非线性增长.例如,MonetDB 存储 LUBM-500 M 所需求的存储量是 LUMB-100 M 的 6.5 倍.而存 50 M 的所需求的存储空间只是 LUMB-10 M 的 4.8 倍.特别对于 LUBM-500 M 数据集,MonetDB 占用的存储空间由小于 TripleBit,转变为 MonetDB 的存储消耗反而超过 TripleBit 约 800 MB.

本文还测试了 TripleBit 存储编码后每个三元组平均所消耗的存储空间、URI 及字符串所需求的存储空间,如表 2 所示.从表 2 数据可以看出,存储每个编码后的三元组平均只需要 5.02~5.97 B.同存储两个整数 ID 所需要的 8B 相比降低了 30%~

Table 2 The Storage of TripleBit

表 2 TripleBit 使用存储空间

Data Set	Bytes Per Triple in Average/B	Storage for URI & Literal/MB
LUMB-10 M	5.02	257
LUMB-50 M	5.48	1 195
LUMB-100 M	5.54	2 396.4
LUMB-500 M	5.59	11 348.7
LUMB-1 B	5.97	22 740.7

37%. 这表明变长整数编码方法是有效的. 表 2 显示 RDF 数据中字符串所占用的存储空间达到 TripleBit 整个存储空间一半以上.

5.3 查询时间比较

在 LUBM-50 M, LUBM-100 M, LUBM-500 M 和 LUBM-1 B 数据集上的查询时间分别如图 3～6 所示. 为了衡量各个系统的整体查询性能,表中还包括查询性能的几何均值. MonetDB 在服务器 1 上不能装载数据集 LUBM-500 M, 因此图 5 只包括 RDF-3X 和 TripleBit 的性能数据.

Queries	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Geo. Mean
# Results	10	10	0	8	1 284	22 001	
Cold Caches							
RDF-3X	0.219 4	0.154 7	0.839 0	0.323 7	13.510 1	0.049 2	0.427 8
MonetDB	4.600 0	3.000 0	6.400 0	1.000 0	19.500 0	0.052 7	2.119 9
TripleBit	0.001 3	0.002 7	0.441 5	0.131 4	12.851 3	0.024 3	0.063 2
Warm Caches							
RDF-3X	0.001 1	0.002 4	0.632 2	0.003 0	2.455 8	0.014 5	0.023 7
MonetDB	0.054 2	0.011 9	6.200 0	0.198 4	1.933 3	0.013 9	0.166 5
TripleBit	0.000 3	0.000 4	0.176 1	0.000 7	1.071 9	0.001 2	0.005 2

Fig. 3 Query time of LUBM-50 M(time in seconds).

图 3 LUBM-50M 的查询时间比较(单位:秒)

Queries	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Geo. Mean
# Results	10	10	0	8	2 528	44 190	
Cold Caches							
RDF-3X	0.226 6	0.159 9	1.813 1	0.366 7	26.669 8	0.070 4	0.596 9
MonetDB	11	5.2	20.5	1.5	55.400 0	0.068 9	4.343 4
TripleBit	**0.196 7**	**0.078 9**	**1.520 2**	**0.161 2**	**19.693 2**	**0.025 6**	**0.352 5**
Warm Caches							
RDF-3X	0.001 1	0.002 6	1.259 0	0.003 1	4.674 2	0.022 0	0.032 4
MonetDB	0.026 8	0.031 1	18.466 7	0.4	0.36	0.028 9	0.200 0
TripleBit	**0.000 3**	**0.000 4**	**0.442 8**	**0.000 8**	**2.125 7**	**0.001 3**	**0.007 0**

Fig. 4 Query time of LUBM-100 M(time in seconds).

图 4 LUBM-100 M 的查询时间比较(单位:秒)

Queries	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Geo. Mean
# Results	10	10	0	8	2 528	219 772	
Cold Caches							
RDF-3X	0.287 7	0.182 4	8.677 6	0.390 0	150.662 9	0.250 6	1.373 2
TripleBit	**0.231 7**	**0.049 3**	**6.635 2**	**0.192 1**	**109.370 2**	**0.026 6**	**0.590 4**
Warm Caches							
RDF-3X	0.001 1	0.002 7	7.324 5	0.003 2	25.693 3	0.098 4	0.074 9
TripleBit	**0.000 3**	**0.000 4**	**2.290 8**	**0.000 9**	**11.980 4**	**0.001 4**	**0.012 7**

Fig. 5 Query time of LUBM-500 M(time in seconds).

图 5 LUBM-500 M 的查询时间比较(单位:秒)

Queries	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Geo. Mean
# Results	10	10	0	8	2 528	439 997	
Cold Caches							
RDF3X	0.306 4	0.337 2	53.396 6	0.361 6	2 335.8	496.197	11.499 2
MonetDB	560.39	562.09	3 081.19	579.29	966.93	4 929.85	1 178.57
TripleBit	**0.147 1**	**0.144 1**	**21.857 1**	**0.231 3**	**69.89**	**99.640 1**	**3.011 7**
Warm Caches							
RDF3X	0.001 8	0.002 9	33.486 1	0.003 5	2 227.59	91.559 5	0.706 9
MonetDB	2.582 4	2.455 2	910.74	6.821 8	95.74	4 608.36	50.895
TripleBit	**0.000 2**	**0.000 3**	**7.559 1**	**0.000 3**	**41.000 3**	**46.163 5**	**0.079 8**

Fig. 6 Query time of LUBM-1 B(time in seconds).

图 6 LUBM-1 B 的查询时间比较(单位:秒)

从图 3～6 中的查询性能数据可以看出,TripleBit 在整体上比 RDF-3X 速度快.对于 LUBM-50 M 数据集,在 Cold Caches 条件下,TripleBit 的平均性能是 RDF-3X 平均性能的 6.8 倍(RDF-3X 的平均性能除以 TripleBit 的平均性能).对于 Q_1,RDF-3X 的查询响应时间是 TripleBit 的 168.8 倍,MonetDB 的查询响应时间是 TripleBit 的 3 538 倍.在 Warm Caches 条件下,TripleBit 的平均性能是 RDF-3X 的 4.6 倍.在 Warm Cache 情况下,TripleBit 执行 Q_1 的速度比 RDF-3X 快 3.7 倍.表 4(LUBM-100 M)、表 5(LUBM-500 M)同样显示 TripleBit 的性能超过 RDF-3X.这是由于在查询的过程中,RDF-3X 在读取一个数据块之前都必须对该数据块进行解压,解压完成之后再进行数据的查找和定位,当读取到需要的数据时,该数据在 CPU 的 Cache 至少经过了两次;而在 TripleBit 中,由于采用的是轻量级的压缩方式,单个三元组的解压与其他的数据无关,因此,在读取到需要的数据时,该数据只读取了一次.

图 3、图 4 的性能数据同样显示 TripleBit 在整体上比 MonetDB 速度快.对于 LUBM-50 M 数据集,在 Cold Caches 条件下,TripleBit 的平均性能是 MonetDB 的 33.6 倍.对于 Q_1,MonetDB 的查询响应时间甚至是 TripleBit 的 3 538 倍.在 Warm Caches 条件下,TripleBit 的平均性能分别是 RDF-3X 和 MonetDB 的 4.6 倍和 32 倍.在 Warm Cache 情况下,TripleBit 执行 Q_1 的速度比 RDF-3X 快 3.7 倍,比 MonetDB 约快 181 倍.可以看出,RDF-3X 和 MonetDB 有更好的缓存管理性能.这是因为 MonetDB 首次运行一个查询时需要将所需的数据调入内存中来进行查找和读取,并且将相应的数据进行缓冲.在第 2 次运行同样的查询时,由于大部分的数据缓存在内存中,MonetDB 在 Warm Caches 的条件下,查询的速度得到了较大的提升.

对于 LUBM-1 B 这样级别的数据集,TripleBit 同样表现出了很高的扩展性.在 Cold Caches 条件下,TripleBit 的查询速度分别是 RDF-3X 和 MonetDB 的 4.5 倍和 555 倍.而在 Warm Caches 条件下,TripleBit 的查询速度表现得更有优势,分别是 RDF-3X 和 MonetDB 的 14 倍和 1 138 倍.特别地,MonetDB 对于中间结果较大的查询,如 Q_6 速度下降得很快.这是由于系统中的缓存有限,在查询的过程中需要的数据被后进来的数据替换掉,但是在下一步操作时被替换的数据又要使用,这使得整个系统频繁调页,导致性能降低.

从图 3～5 可以看出,在 Warm Cache 情况下,TripleBit 在 3 个数据集上执行 Q_1,Q_2,Q_4,Q_6 的性能基本保持不变.这一方面说明 TripleBit 的存储组织使得回答 Q_1,Q_2,Q_4,Q_6 的相关数据存储在较小的范围;另一方面也说明了 TripleBit 的性能稳定.

在执行查询过程中,记录了各系统的内存使用情况.由于 Q_6 的中间结果较大,内存使用情况变化明显.在实验中,记录了各系统在 LUBM-100 M 和 LUBM-1 B 上执行 Q_6 的内存峰值,如图 7 所示,在虚拟内存的使用上,TripleBit 小于 RDF-3X 和 MonetDB 的使用量.在物理内存的使用上,TripleBit 介于两者之间.可以看出 TripleBit 产生的中间结果较小,执行查询过程中加载数据量小.

Data set	LUBM-100 M		LUBM-1 B	
Memory	Virtual Mem.	Phy. Mem.	Virtual Mem.	Phy. Mem.
TripleBit	5.545	3.7	47.8	33
RDF-3X	8.249	1.6	84.1	18
MonetDB	10.5	8.1	89.3	61

Fig. 7 Peak memory usage when executing LUBM Q_6 (GB).

图 7 执行 LUBM-Q_6 时内存使用峰值(单位:GB)

6 结束语

本文介绍了具有高扩展的 RDF 数据存储系统 TripleBit. 针对 URI 和 ID,给出了不同的压缩存储方法. 既降低了存储空间,在读取数据的速度上也有提升. 在存储方式上,采用了紧凑化的存储方法,将数据进行分片,进一步提高了系统在读取数据时的吞吐率,为快速处理查询奠定了基础. TripleBit 采用了轻量级的查询计划动态生成算法. 实验表明, TripleBit 具有良好的扩展性,能够应对大规模 RDF 数据的存储需求.

参 考 文 献

[1] World Wide Web Consortium: RDF/XML Syntax Specification (Revised) [OL]. [2004-02-10]. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar

[2] W3C SWEO Community Project. Linking open data on the semantic Web [OL]. [2012-03-17]. http://www.w3.org/wiki/SweoIG/TaskForces/Community-Projects/LinkingOpenData

[3] Du Xiaoyong, Wang Yan, Lü Bin. Research and development on semantic Web data management [J]. Journal of Software, 2009, 20(11): 2950-2964 (in Chinese)
(杜小勇, 王琰, 吕彬. 语义 Web 数据管理研究进展[J]. 软件学报, 2009, 20(11): 2950-2964)

[4] Broekstra J, Kampman A, Harmelen F. Sesame: A generic architecture for storing and querying RDF and RDF schema [G] //LNCS 2342: Proc of the 1st Int Semantic Web Conf. Berlin: Springer, 2002: 54-68

[5] Weiss C, Karras P, Bernstein A. Hexastore: Sextuple indexing for semantic Web data management [C] //Proc of VLDB'2008. Trondheim, Norway: VLDB Endowment, 2008: 1008-1019

[6] Neumann T, Weikum G. Scalable join processing on very large RDF graphs [C] //Proc of ACM SIGMOD 2009. New York: ACM, 2009: 627-639

[7] Neumann T, Weikum G. The RDF-3X engine for scalable management of RDF data [J]. VLDB Journal, 2010, 19(1): 91-113

[8] Abadi D J, Marcus A, Madden S R, et al. Scalable semantic Web data management using vertical partitioning [C] //Proc of VLDB'2007. Trondheim, Norway: VLDB Endowment, 2007: 411-422

[9] SWAT Projects of Lehigh University. LUBM [OL]. [2012-03-17]. http://swat.cse.lehigh.edu/projects/lubm/.

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

[11] Wilkinson K, Sayers C, Kuno H, et al. Efficient RDF storage and retrieval in Jena2 [C] //Proc of the 1st Int Workshop on Semantic Web and Databases. 2003: 131-150

[12] Wilkinson K. Jena property table implementation [C] //Proc of the 2nd Int Workshop on Scalable Semantic Web Knowledge Base Systems. 2006: 35-46

[13] Chong E I, Das S, Eadon G, et al. An efficient SQL based RDF querying scheme [C] //Proc of VLDB'2005. Trondheim, Norway: VLDB Endowment, 2005: 1216-1227

[14] Agrawal R, Somani A, Xu Y. Storage and querying of E-commerce data [C] //Proc of VLDB'2001. Trondheim, Norway: VLDB Endowment, 2001: 149-158

[15] Beckmann J, Halverson A, Krishnamurthy R, et al. Extending RDBMSs to support sparse datasets using an interpreted attribute storage format [C] //Proc of ICDE'2006. Los Alamitos, CA: IEEE Computer Society, 2006

[16] Janik M, Kochut K. BRAHMS: A workbench RDF store and high performance memory system for semantic association discovery [G] //LNCS 3729: Proc of the 4th Int Semantic Web Conf. Berlin: Springer, 2005: 431-445

[17] Abadi D J, Marcus A, Madden S R, et al. SW-Store: A vertically partitioned DBMS for semantic Web data management [J]. The VLDB Journal, 2009, 18(2): 385-406

[18] Abadi D J. Column stores for wide and sparse data [C] //Proc of the 3rd Biennial Conf on Innovative Data Systems. 2007: 292-297

[19] Sidirourgos A, Goncalves R, Kersten M, et al. Column store support for RDF data management: Not all swans are white [C] //Proc of VLDB'2008. Trondheim, Norway: VLDB Endowment, 2008: 1553-1563

[20] Bonstrom V, Hinze A, Schweppe H. Storing RDF as a graph [C] //Proc of the first Latin American Web Congress. Piscataway, NJ: IEEE, 2003: 27-36

[21] Kim Y H, Kim B G, Lee J, et al. The path index for query processing on RDF and RDF schema [C] //Proc of the 7th Int Conf on In Advanced Communication Technology. Piscataway, NJ: IEEE, 2005: 1237-1240

[22] Matono A, Amagasa T, Yoshikawa M, et al. A path-based relational RDF database [C] //Proc of 16th Australasian Database Conf. New York: ACM, 2005: 95-103

[23] Wu G, Li J, Wang K. System Π: A hypergraph based Native RDF repository [C] //Proc of WWW'2008. New York: ACM, 2008: 1035-1036

[24] Yan Y, Wang C, Zhou A, et al. Efficient indices using graph partitioning in RDF triple stores [C] //Proc of ICDE'2009. Los Alamitos, CA: IEEE Computer Society, 2009: 1263-1266

[25] Zou L, Mo J, Chen L, et al. gStore: Answering SPARQL queries via subgraph matching [C] //Proc of VLDB'2011. Trondheim, Norway: VLDB Endowment, 2011: 482-493

[26] Altra M, Chaoji V, Zaki M, et al. Matrix: "Bit" loaded: A scalable lightweight join query processor for RDF data [C] // Proc of WWW'2010. New York: ACM, 2010: 41-50

[27] Vidal M, Ruckhaus E, Lampo T, et al. Efficiently joining group patterns in SPARQL queries [C] //Proc of the 7th Extended Semantic Web Conf. Berlin: Springer, 2010: 228-242

[28] Lü Bin, Du Xiaoyong, Wang Yan. SPARQL query optimization based on property correlations [J]. Journal of Computer Research and Development, 2009, 46(Suppl.): 119-125 (in Chinese)
(吕彬,杜小勇,王琰. 基于属性相关的 SPARQL 查询优化方法[J]. 计算机研究与发展, 2009, 46(增刊): 119-225)

[29] Abadi D J, Madden S R, Hachem N. Column-stores vs. row-stores: How different are they really?[C] //Proc of ACM SIGMOD'2008. New York: ACM, 2008: 967-980

Yuan Pingpeng, born in 1972. Associate professor in the School of Computer Science and Technology at Huazhong University of Science and Technology. Senior member of China Computer Federation and member of ACM and IEEE. His research interests include semantic Web, database etc.

Liu Pu, born in 1987. Received his MSc degree in computer architecture from Huazhong University of Science and Technology (HUST) in 2012. His research interests include semantic Web and sematic data storage system.

Zhang Wenya, born in 1988. Receive her bachelor's degree in computer science from Huazhong University of Science and Technology, Hubei, China, in 2011. Since 2011, master candidate in computer system structure from HUST. Her current research interests include semantic Web technology and massive data processing.

Wu Buwen, born in 1986. PhD candidate in computer architecture of Huazhong University of Science and Technology. His research interests include semantic Web and massive data processing.

附录 A LUBM 数据集的查询语句.

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>

Q_1: SELECT ?x WHERE { ?x rdf:type ub:ResearchGroup. ?x ub:subOrganizationOf <http://www.Department0.University0.edu>. }

Q_2: SELECT ?x WHERE { ?x rdf:type ub:FullProfessor. ?x ub:name ?y1. ?x ub:emailAddress ?y2. ?x ub:telephone ?y3. ?x ub:worksFor <http://www.Department0.University0.edu>. }

Q_3: SELECT ?x ?y ?z WHERE { ?x rdf:type ub:UndergraduateStudent. ?y rdf:type ub:University. ?z rdf:type ub:Department. ?x ub:memberOf ?z. ?z ub:subOrganizationOf ?y. ?x ub:undergraduateDegreeFrom ?y. }

Q_4: SELECT ?x ?y WHERE { ?x rdf:type ub:GraduateStudent. ?y rdf:type ub:GraduateCourse. <http://www.Department0.University0.edu/AssociateProfessor0> ub:teacherOf ?y. ?x ub:takesCourse ?y. }

Q_5: SELECT ?x ?y ?z WHERE { ?y ub:teacherOf ?z. ?x rdf:type ub:UndergraduateStudent. ?x ub:advisor ?y. ?z rdf:type ub:Course. ?x ub:takesCourse ?z. ?y rdf:type ub:FullProfessor. }

Q_6: SELECT ?x ?y WHERE { ?x ub:worksFor ?y. ?x rdf:type ub:FullProfessor. ?y ub:subOrganizationOf <http://www.University0.edu>. ?y rdf:type ub:Department. }