基于语义的通用数据抽取方法

张建英 孙永洁 王秀坤
(大连理工大学计算机科学与工程系 大连 116024)

摘 要 关系数据库可以看作是元组以及外键关系构成的有向图。为便于数据复制以及共享,在进行数据抽取时,往往既要使语义上相关的数据一起抽取,又要使得抽取的数据尽量逻辑上独立。将多根树作为语义上相关、逻辑独立的数据集,给出了关系数据抽取方法并进行了实现。在Oracle中,使用TPC-C数据库结构对该方法进行了测试与分析,从而验证了算法的有效性和通用性。

关键词 多根树,语义相关,数据抽取
中图法分类号 TP311.13 **文献标识码** A

Generic & Semantic-based Data Extraction Approach

ZHANG Jian-ying SUN Yong-jie WANG Xiu-kun
(Department of Computer Science and Engineering, Dalian University of Technology, Dalian 116024, China)

Abstract A relational database can be viewed as a directed graph constructed by tuples and foreign key references. To facilitate data replication and sharing, semantic-relativity and logically independence should be satisfied when relational data is extracted. Multi-tree structures are employed as clusters of such data extracted from a relational database in this paper. Then the corresponding data extraction approach was proposed and implemented. We evaluated the extraction algorithm on a TPC-C database in Oracle, demonstrating the effectiveness and generalization of the approach.

Keywords Multi-tree, Semantic-related, Data extraction

数据抽取技术在信息处理领域有着重要的作用,不论是系统之间还是内部子系统之间都会经常碰到数据抽取、复制、共享的问题。数据抽取是数据复制的前提,抽取后的数据可以加载到数据仓库中,也可以作为交换的数据给别的系统共享。目前数据抽取技术研究的热点是非结构化、半结构化的Web数据抽取[1-3]。结构化数据抽取工具主要是各大厂商的ETL[4]工具和部分中间件,这些工具大都侧重单表抽取,抽取时还要进行转换、加载等处理。很多应用系统内也有自己的关系数据抽取模块,但是这些模块大都与数据库模式有关,在不同的应用间不通用。本文为结构化数据提供一种与模式无关的、基于语义的数据抽取方法。

结构化数据之间最重要、最常见的关系就是通过外键构成的参照关系。通常由这些参照连接起来的数据是个有向图,但图理解和使用都比较复杂。用树形结构来表示数据间的关系由来已久,例如,传统的层次模型、后来的XML,大体上都是基于树结构的。但是这种传统的单根树(只有一个根节点)结构不能很好地表示多对多的关系,例如,家庭的双亲关系就无法用单根树表示。陈有祺引入多根树(Multi-Roots Tree)[5]来解决这样的问题,George W. Furnas 与 Jeff Zacks 提出的 Multitree[6] 实际上是相同的。陈有祺的多根树的定义如下。

在无回路的有向图 G 中,若它的任何两个节点之间至多有一条通路,则称 G 为多根树,G 中所有入度为0的节点为 G 的根,所有出度为0的节点为 G 的叶。

实际上多根树思想在计算机领域已有很多的应用。我们所熟知的C++语言的多根继承机制、JAVA语言的多接口实现等都是多根树思想的应用。

本文首先回顾了数据库的模式图、数据图,并将模式图、数据图看作或者转化成多根树,以便利用多根树思想来实现基于语义的数据抽取。接着在对语义完整性、数据库完整性进行了阐述的基础上,给出了语义相关抽取算法。最后,实现了该算法,并验证了算法的有效性。

1 语义相关的数据抽取

1.1 模式图、数据图以及多根树

数据库由模式和相应的数据构成,可以分别看作为模式图和数据图[7]。模式图是由数据库模式产生的图,每一个关系看作一个节点,关系间的参照关系为边。这些边是有向的,方向由参照关系指向被参照关系(有时也采用相反的表示方法),被参照节点可以看作参照节点的父亲节点。数据图是由元组数据构成的图,关系的一行元组数据是一个节点,元组间的参照为边,方向与它们所在的关系模式的参照方向一致。容易看出,数据图是依据模式图而来的,数据图中的节点是模式图中节点所包含的元组,数据图中的边是由模式图中外键

到稿日期:2009-05-14 返修日期:2009-08-14 本文受国家自然科学基金(60873054)资助。
张建英(1973—),男,博士生,讲师,主要研究方向为分布式数据库技术,E-mail:zhangjy@dlut.edu.cn;孙永洁 硕士生,主要研究方向为分布式数据库技术、数据库系统;王秀坤 教授,博士生导师,主要研究方向为数据库系统、分布式数据库技术。

关系决定的。

显然，不论是模式图还是数据图都是有向图，都会存在着回路以及菱形[6]。因此严格地说，这样模式图或数据图并不是多根树。但是，关系间通过一对多的外键参照，大体上，关系之间是层次结构的；即使存在局部的回路、菱形，也不妨碍对整体结构的理解。在进行数据库设计时，对于模式图，尽量不要含有回路、环、菱形结构。这样在生成的数据图中，就不会出现这样的结构。即使如此设计，有些回路、菱形还是不能完全消除，还需特别处理。一方面，可以修正多根树的定义，使其可以允许菱形结构[6]。另一方面，可以在对这些含回路的"多根树"操作前，进行预处理，临时禁用回路上某个外键参照关系；操作完成后，再恢复该外键关系。将有向图近似成多根树不是本文的重点，这里不再赘述。

如果每个关系最多只有一个外键参照关系，则单根树结构可以表示，但是对于关系有多个外键的情形，单根树就不适合了，多根数是个自然的选择。综上，数据图可以被认为是棵多根树，那么我们就能以多根树为基础，进行语义相关的数据抽取了。

1.2 语义相关

关系型数据库本质上就是一个语义相关的数据集合，关系之间的参照就是这种语义的体现。这样的参照关系，将不同关系的数据紧密的关联在一起。数据抽取应考虑抽取数据的语义完整性，应该围绕被抽取目标数据将语义相关的数据作为一个整体提取出来。给定的抽取参数是个数据元组，或是限定在一个或多个特定关系上的一个条件语句对应的一组数据节点，抽取就是围绕这些抽取节点展开的。究竟抽取什么样的数据取决于其它数据与这些数据的语义相关性。对于多根树中两个节点 a 与 b，若 a 是 b 的祖先节点或子孙节点，则说节点 a 与 b 语义相关。

抽取出的数据是原数据库多根树的一棵多根子树。首先，所有被抽取的数据的父节点都应被抽取到结果中，也就是任何节点的祖先必需是完整的。这样，在任何时候都能使得这些数据满足参照完整性要求。另外，对于抽取节点以下的所有节点，都应该出现在结果的多根树中，目的是满足抽取数据的完整性，即所有的抽取节点数据的子孙节点数据都要被抽取。需要强调的是，这些子孙节点的其它祖先节点也要被抽取。虽然它们不是与抽取节点语义相关的节点，但是考虑到数据库的参照完整性，我们也要将这样的节点包含在内，后面的抽取算法中也是这样实现的。

图 1(a)为给定一棵多根树，右侧图 1(b)为以 $d1$ 为控制节点经过算法抽取的结果多根树，即与 $d1$ 语义相关的节点集。图中元组 $a1$，$a2$ 来自关系 A；$b1$，$b2$ 来自关系 B……。

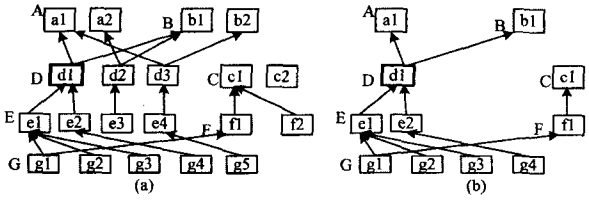

图 1 多根树抽取

2 数据抽取算法

在数据库的多根树中，给定一个数据抽取条件，这个条件往往是一个控制节点或者控制节点集，以其为参数得到结果多根树。抽取的过程就是将与该控制节点语义相关联的节点全部抽取出来。

首先给出几个符号的定义，F 表示控制节点集。R 表示多根树。$\Gamma_F(R)$表示在多根树 R 上基于条件 F 的抽取结果。算法任务是抽取与控制节点集的所有语义相关节点。

我们的抽取算法的目的是抽取所有相关的节点到新的多根树中，结果要满足下面 3 个要求。

(1) 对于每一个节点，所有的祖先节点都要包含在结果中。

(2) 对于控制节点，它的所有子孙节点都要包含在结果中。

(3) 不要有重复的节点插入(即使当"多根树"中有回路和菱形的存在时)。

2.1 基于一个控制节点的抽取算法

本节先讨论基于一个控制结点的数据抽取算法，在下节讨论更为一般的数据抽取算法。

```
输入:一个控制节点 f,初始多根树 R
输出:多根树 ResultMT
变量定义:UpNodes:控制节点的祖先数据节点集
         DownNodes:控制节点的孩子节点集
         DiscoveredNodes:已经抽取出来的节点集
         tempNode:一个临时节点
//初始化
UpNodes←{f};DownNodes←{f};
DiscoveredNodes←{f};
//当多根树的控制结点下部或上部还有没有遍历的节点时,循环处理
While UpNode≠Φ|| DownNodes≠Φ do
  if UpNodes≠Φ then
    tempNode←fetchOneNode(UpNodes);
    if tempNode∉DiscoveredNodes then
      DiscoveredNodes←DiscoveredNodes∪{tempNode};
      ResultMT←ResultMT∪{tempNode };
      //对于控制结点上部的结点,还要递归地向上遍历
      UpNodes←UpNodes∪getParentNodes(tempNode);
  end if;
end if;
if DownNodes≠Φ then
    tempNode←fetchOneNode(DownNodes);
    if tempNode∉DiscoveredNodes then
      DiscoveredNodes←DiscoveredNodes∪{tempNode};
      ResuletMT← ResuletMT∪{tempNode};
      //对控制结点下的结点,即要向下又要向上遍历
      DownNodes←DownNodes∪getChildNodes(tempNode);
      UpNodes← UpNodes∪
      getParentNodes(tempNode);
    end if;
  end if;
end;
return ResultMT;
```

对于模式图中的回路问题，在上节提出采用人工干预的方法。对于菱形问题，使用 DiscoveredNodes 记录已经抽取出来的节点，用来避免有菱形时节点被多次添加的情况；当新节点添加到结果多根树时，先确定 DiscoveredNodes 是否已经包含该节点，如果未包含则添加该节点。另外，在关系数据库

中实现该算法时，使用 ODBC 获取元数据信息使得算法与具体的数据库管理系统、关系模式无关，从而使得算法通用。算法中多数操作可以基于关系代数，即以关系为处理单位，因此能充分利用 SQL 语言来实现。

2.2 基于控制节点集数据抽取算法

更一般的数据抽取是采用控制节点为集合的方式。在下面的算法中，$\Gamma n(R)$调用为上节的单控制节点抽取算法。

输入：控制节点集 $F=\{\{fi1,fi2,\cdots\},\cdots,\{fi1,fi2,\cdots\},\cdots,\{\cdots\}\}$。其中$\{fi1,fi2,\cdots\}$表示在结果多根树的节点应该被所有的条件共同控制。基于控制节点集 F 抽取的最终的多根树是基于控制节点集$\{fi1,fi2,\cdots\}$抽取的所有的多根树的并集。

初始多根树：R

输出：结果多根树 ResultMT

```
ResultMT←Φ;
While F≠Φ do
  tempNodeSet←fetchOneElement(F);
  foreach node∈tempNodeSet do
    if tempMultitree=Φ then
      tempMultitree←Γn(R);
    else
      tempMultitree← tempMultitree ∩Γn(R);
    end if;
  end for;
  if ResultMT=Φ then
    ResultMT←tempMultitree;
  else
    ResultMT←ResultMT∪tempMultitree;
  end
    tempMultitree←Φ;
  end
  return ResultMT;
```

3 测试及评价

采用事务处理性能协会(Transaction Processing Performance Council，简称 TPC)提供的 TPC-C 标准数据库来验证上述算法。一家销售供应商，拥有一批分布在不同地方的仓库和地区分公司。通常每个仓库供货覆盖 10 家地区分公司，每个地区分公司服务 3000 名客户。公司共有 100,000 种商品，分别储存在各个仓库中。仓库、地区分公司、客户等的关系如图 2 所示。在 Oracle 数据库上实现该数据库，目的主要是验证算法的有效性，而非进行性能评估。为了简化实现，在数据库中只生成了一个仓库的数据，在普通微机上进行了实现。

算法执行前利用图的深度优先搜索检测是否有回路存在。如果检测到回路，则进行预处理，方法在前面已经提及。图 2 数据库模式图中存在菱形，由于算法中 DiscoveredNodes 的使用，菱形不影响本文算法的正确性。

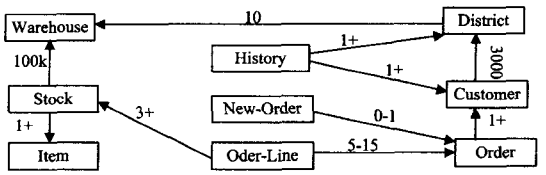

图 2 TPC-C 数据库模式图

表 1 描述的是抽取指定表格内一行元组的所有语义相关元组的所需时间。从表中可以看出抽取时间的大小主要取决于抽取结果的大小和抽取指定关系的复杂程度，即祖先节点数、后代节点数，后代结点引用的是祖先结点。抽取结果量越大，抽取关系越复杂越多，消耗时间越多。表 1 给出的消耗时间仅具有参考意义，算法实现时，包含数据库的 I/O 时间及结果输出时间。还需说明的是，我们没能查阅到类似的算法实现，所以没有给出该算法与其它算法的性能比较。

表 1 抽取所需的时间

表名	外键个数	祖先节点数	后代节点数	后代结点其它祖先	抽取结果(k)	消耗时间(s)
Warehouse	0	0	8	0	29081	175
District	1	1	5	2	16281.6	11.84
Customer	1	2	4	2	5.57	0.79
History	2	3	0	0	1.00	0.21
Order	1	3	2	2	5.42	0.77
New-Order	1	4	0	0	0.79	0.17
Order-Line	2	5	0	0	1.31	0.25
Stock	2	2	1	3	2.29	0.56
Item	0	0	2	4	4.20	0.71

抽取后数据的存储格式、如何加载是与数据抽取密切相关的两个问题。将抽取的数据按照拓扑顺序(从根到叶子节点)生成文件。导入时只需按行读取，对于每行数据(包含表名，各列数据值)，生成 Insert 语句执行即可。对于冲突，即目标数据库中已存在该条记录，系统会提示是要覆盖原值还是跳过保留原值。

这种基于多根树思想的数据抽取有很多优点。

首先是与模式无关，与数据库平台无关。该算法应用广泛，适用各个应用系统的表结构，只要这些系统是通过外键参照关系被很好设计的。另外，通过使用 ODBC 可适用于各种数据库平台。

其次是数据完整性，数据语义更加健壮，一次抽取便利了以后多种操作。例如对于 Order 数据抽取，以前只抽取 Order 以及 Order 参照的 New-Order，Order-Line，当一个应用要再查询该 Order 涉及到了哪个(些)Warehouse 时，又要重新抽取数据。现在我们的算法一次将语义相关的数据完整抽取，这一点在数据分布、共享中非常重要。

在实际应用中，利用该算法可以完成例如方案下发、计划上报等数据抽取、导入操作，只需给定方案的编号，就能实现相关信息的完全抽取。另外，该数据抽取算法也适合应用于数据分布，可以尽可能实现相关数据紧密存放，便于查询的本地化，减少跨站点的连接操作。

结束语 本文引入多根树的思想，研究了基于语义的数据抽取方法，给出了一种通用的数据抽取算法，并且进行了实现和测试。

这种语义数据抽取的特点有：

(1)抽取过程中着重考虑数据的语义完整性和参照完整性，以实现逻辑独立。

(2)屏蔽了不同数据库系统、不同数据库模式间的差别。

(3)可以用于 ETL 工具实现、数据分布等方面。

参 考 文 献

[1] Laender A H, Ribeiro-Neto B A, da Silva A S, et al. A brief survey of web data extraction tools[J]. SIGMOD Rec., 2002, 31(2): 84-93

[2] Zhai Y, Liu B. Web data extraction based on partial tree alignment[C]// Proceedings of the 14th international Conference on World Wide Web WWW '05. ACM, New York, NY, 2005: 76-85

[3] Gottlob G, Koch C, Baumgartner R, et al. The Lixto data extrac-

tion project: back and forth between theory and practice[C]// Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (Paris, France, June 2004). PODS '04. ACM, New York, NY, 2004: 1-12

[4] Vassiliadis P, Simitsis A, Skiadopoulos S. Conceptual modeling for ETL processes[C]// Proceedings of the 5th ACM international Workshop on Data Warehousing and OLAP (McLean, Virginia, USA, November 2002). DOLAP '02. ACM, New York, NY, 2002: 14-21

[5] 陈有祺. 多根树[J]. 南开大学学报: 自然科学版, 1989(22): 26-28

[6] Furnas G W, Zacks J. Multitrees: enriching and reusing hierarchical structure[C]// Proceedings of the SIGCHI conference on Human factors in computing systems. 1994: 330-336

[7] Hristidis V, Papakonstantinou Y. Discover: Keyword Search in Relational Databases[C]// the 28th VLDB Conference. Hong Kong, China, 2002: 670-681

(上接第 177 页)

化 SPL 流程模型。实例化后的 SPL 流程模型将驻留在 SPL 执行引擎系统中，为具体流程实例的创建提供支持；3）根据 SPL 流程模型创建 SPL 流程实例。

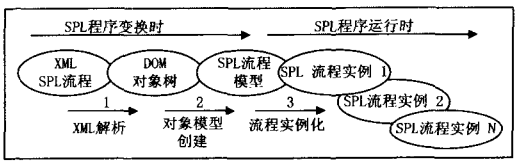

图 6 语义流程变换

流程模型通过语义服务适配器、规则服务适配器、数据适配器动态地将服务和规则组合起来，以支持流程执行。

4.2 语义程序的部署变换

SPL 程序执行变换后生成可执行程序，并被部署到 SPL 执行引擎中，SPL 程序只有通过部署变换后才能被其他的 SPL 程序或应用系统以服务的方式调用。

语义程序的部署变换包含 Web 服务变换和语义 Web 服务变换，部署变换过程就是实现 SPL 可执行程序的服务化，包括 Web 服务化和语义 Web 服务化。Web 服务变换将 SPL 可执行程序变换成对应的 Web 服务的 WSDL 文档，而后者则将 SPL 可执行程序变换成对应的语义 Web 服务，并将其服务描述文档发布到互联网上，SPL 程序的客户端可以采用标准的 Web 服务或语义 Web 服务方式调用，SPL 程序部署变换过程如图 7 所示。

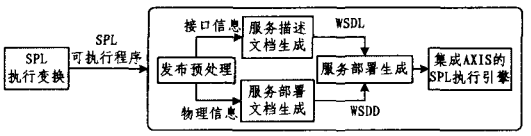

图 7 SPL 程序部署变换

SPL 程序部署变换包括 WSDL 服务描述文档生成、服务部署描述文档生成和服务部署 3 个主要部分。预处理部分获取经过变换后的 SPL 程序的所有对象，根据其中的 SPL 程序对外接口信息，通过服务描述文档生成 SPL 程序对应的 Web 服务的 WSDL 文档，并结合执行引擎物理地址、端口信息等物理信息，通过服务部署文档生成 WSDD 文档，从而将两个文档一并交给服务部署模块后部署在 AXIS 上，以便 SPL 程序运行时能调用 Web 服务访问程序中的规则。

结束语 在语义编程语言 SPL 基础上，提出了一种面向语义 Web 服务的语义程序变换方法。基于语义程序的语法结构和语义信息，该方法不仅给出了语义数据类型、语义规则、语义服务和语义流程等语义信息的执行变换，而且还给出了语义程序到 Web 服务的部署变换。该方法不但能有效地提高面向服务程序设计的灵活性和健壮性，还有助于提高业务流程的柔性和重用性。未来的研究工作是进一步完善本文中的语义程序变换方法，并研究其形式化方法和开发支撑工具，以形成一套完整的、更具有实用价值的语义程序变换体系。

参 考 文 献

[1] Burstein M, Bussler C, Zaremba M, et al. A Semantic Web Services Architecture[J]. IEEE Internet Computing, 2005, 9(5): 72281

[2] 曹虹华，应时，杜德慧，等. 一种面向语义 Web 服务的软件设计语言和设计方法[J]. 电子学报, 2007, 35(12A)

[3] Web Services Business Process Execution Language Version2. 0 [OL]. http://docs. oasis-open. org/wsbpel/2. 0/CS01/wsbpel-v2. 0-CS01. pdf, January 2007

[4] Web Services Choreography Description Language Version 1. 0 [OL]. http://www. w3. org/TR/2005/CR-ws-cdl-10-20051109/, November 2005

[5] Rosenberg F, Dustdar S. Business Rules Integration in BPEL-A Service-Oritented Approach[C]// 7th Internatinal IEEE Conference on E-Commerce Technology(CEC2005). Munich Germany

[6] Karastoyanova, Houspanossian D, Cilia A, et al. Extending BPEL for Run Time Adaptability[C]// Proc. of the 9th International Enterprise Distributed Object Computing Conference (EDOC 2005). Enschede, The Netherlands, September 2005

[7] Kang Zuling, Wang Hongbing, Hung P K. WS-CDL+: An Extended WS-CDL Execution Engine for Web Service Collaboration

[8] Mendling J, Hafner M. From WS-CDL Choreography to BPEL Process Orchestration[J]. Journal of Enterprise Information Management (JEIM), 2008, 21(5)

[9] Norton B, Pedrinaci C, Omingue J. Semantic Execution Environments for Semantics-Enabled SOA

[10] Nitzsche J, van Lessen T, Karastoyanova D, et al. BPEL for Semantic Web Services(BPEL4SWS)[C]// On the Move to Meaningful Internet Systems 2007: OTM 2007 workshops, ser. LNCS. vol. 4805/2007, Springer, 2007: 179-188

[11] RO4SSOA. Reference Ontology for Semantic Service Oriented architectures[OL]. http://docs. oasis-open. org/ex-semantics/sematicsoaro/0. 1, September 2007

[12] van Lessen, Nitzsche T, Dimitrov J, et al. An Execution engine for BPEL4SWS[C]// 2nd Workshop on Business Oriented Aspects concerning Semantics and Methodologies in Service-oriented Computing (SeMSoc) in conjunction with ICSOC. Vienna, Austria

[13] Margaria T. The Semantic Web Services Challenge: Tackling Complexity at the Orchestration Level[C]// Proc. ICECCS'08, 13th IEEE Int. Conf. on Engineering of Complex Computer Systems. Belfast (UK), April 2008

[14] Cao Honghua, Ying Shi, Cui Hua, et al. Towards a Framework for Designing and Executing Business Process Based on Semantic Web Services[C]// International Conference on Wireless Communications, Networking and Mobile Computing. 2008: 1-4

[15] Drools. Drools Boot Camp[EB/OL]. http://www. jboss. org/drools/, 2009