一种轻型关系型数据源包装器的设计

A Lightweight Wrapper System for Relational Data Sources

李效东　顾毓清

（中国科学院软件所　北京 100080）

Abstract　XML is emerging as a de facto standard for data representation and exchange on the Web. However, currently and in the forseeable future, most data will continue to be stored in and retrived from relational database systems. It is mainly decided by the reliabilty, maturity, efficience and lots of tools with RDBMS. Many challenges exist as converting flat, normalized relational data into hierachical, unnormalized XML data. Based on XML technologies, the paper addresses a novel and lightweight approach to translate XML queries into SQL requests and transforms the retured tuples into XML representation in the context of data integration system. A wrapper for relational soruces was developed based on the approach. The paper is an important part of the research on integrated query processing over hetergeneous data sources.

Keywords　XML query, Wrapper, Data integration system, Relational data sources

包装器是自治异构数据源集成系统中的重要组成部分。随着与 XML[1]有关的标准不断制定和完善，越来越多的数据被用 XML 表示，同时必须注意的一个事实是，目前和在一个可以预见的未来，大部分应用系统，甚至是新的基于 Web 的应用系统，仍然将关系数据库系统作为数据存储和查询的首选。关系数据库系统的可靠性、技术的成熟性、丰富的工具、高性能等，都决定了这一点。本文探讨在自治异构数据源集成系统情境下，如何将 XML 查询翻译成 SQL 查询，如何将关系数据转换为 XML 数据表示。

1. 预备知识

本小节，首先介绍本论文的研究背景和情境(context)。

1.1　基于半结构化数据模型的数据集成系统

数据集成的需求由来已久，对数据集成系统的研究一直是数据库研究领域一个较热门的课题[2]。近年来随着 Web 逐渐成为信息服务的主导平台，对 Web 环境下的数据集成系统的研究也越来越成蓬勃发展的趋势[2~4]。

数据集成系统也被称为信息集成系统、中介系统或信息搜集代理等。数据集成系统为多个自治异构数据源的查询提供一个统一界面，方便用户进行集成查询。它自动地将用户的查询请求分解为针对各个数据源的查询请求，然后将各个查询结果合并成最终的结果呈现给用户。多个数据源的存在对用户来说是透明的，用户好像只对一个单一数据源进行操作。

图 1 是一个数据集成系统的原型结构示意图。从抽象层次上来说，一个数据集成系统主要由两部分组成，查询处理器和包装器。前者负责用户查询请求的识别、分析、分解、处理，以及合并查询结果等，而后者主要负责将集成系统的查询翻译成数据源能够理解的查询，并将数据源返回的查询结果转换成集成系统能够理解的数据格式。每一个包装器系统随数据源的不同而不同，有针对 Web 文档的半结构化数据源包装器[5]，也有针对传统数据库系统的结构化数据源包装器[6]。本文的目的在于设计一种关系型数据库系统的包装器。

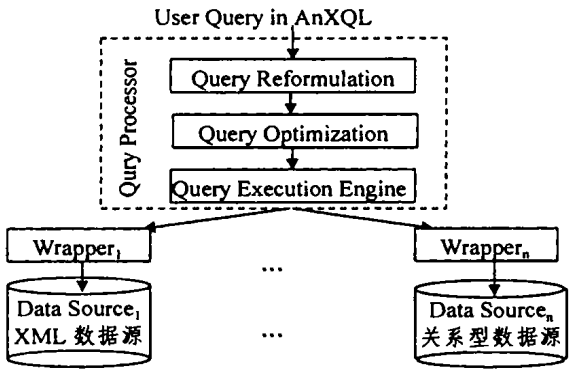

图 1　数据集成系统原型结构图

传统的数据集成系统主要基于结构化数据模型研究和设计[7]，由于 Web 数据源具有半结构化的特点，基于半结构化数据模型构造数据集成系统的研究成为近几年数据库研究界的一个热点[2]。作者在文[8]中研究了基于 XML 这种半结构化数据模型构造数据集成系统的若干问题。其中，用一种 XML 查询语言 AnXQL 来表达用户查询，与某数据源对应的包装器负责将 AnXQL 查询翻译成数据源能够理解的查询形式，并且将返回的该数据源特定的数据格式转换为 XML 数据表示。

1.2　XML 数据模型与 AnXQL 查询语言

AnXQL 查询语言是作者在文[8]中设计的一种 XML 数据查询语言。本部分我们将对它做简单介绍，详细内容及语法表示请参考文[8,9]。我们对文[10]提出的数据模型和查询语言进行必要的扩展以适应表示 XML 数据。先给出构造 AnXQL 查询语言至关重要的几个定义如下：

定义 1(边加标签的带根有向图)　一个带根、边加标签有向图是一个四元组 $G=(V,E,r,\lambda)$。其中 V 是一个非空的结点集合；$E\subseteq V\times V$ 是图中边的集合；(V,E)代表一个有向

李效东　中科院软件所在读博士生，专注于与数据管理技术相关的领域的研究，涉及 XML、半结构化数据、数据集成、查询处理及优化等。**顾毓清**　研究员，博导。

多重图(directed multi-graph);$r \in V$ 表示根,并且满足,任意一个 V 中的节点,从 r 开始经过 E 中的边都能够到达;$\lambda: E \to Label$ 是一个函数,表示边集合 E 到标签集合 Label 的映射。

定义 2(XML 数据模型 XTree) 论域 D 是包括所有字符串类型数据的集合,D 上的所有带根、边加标签有向图组成的集合用 T 来表示 $T \subseteq G$,T 中的元素 t,要么是一个原子数据,即有 $t=d \in D$;要么为 $t=d[t_1,\cdots,t_n]$,其中 $d \in D, t_1,\cdots,t_n \in T$,称 d 为标签(label),$t_1,\cdots,t_n$ 为 t 的子树。$t=d[t_1,\cdots,t_n]$ 对应到 XML 文档数据,称 t 为 XML 树,简称 XTree,它有如下属性:

· 树的每一个节点都有一个唯一的标识符(ID)。这个标识符可以显式地以 XML 文档某一元素的 ID 属性来标识,也可以为其分配一个唯一的 ID 来标识;

· 以 XML 的术语,t 为元素(element),非叶标签 d 为元素类型(element type),$t_1,\cdots,t_n$ 为 t 的子元素(subelements);叶标签 d 为原子对象时,为文字值或空元素;

· 当考虑 $[t_1,\cdots,t_n]$ 中各子树的顺序时,称为顺序的 XTree(ordered XTree)模型,当不考虑 $[t_1,\cdots,t_n]$ 中各子树的顺序时,称为无顺序的 XTree(unordered XTree)模型。

定义 1 是定义 2 的基础,XML 查询语言在 XML 数据模型 XTree 上构造。路径表达式(path expression)是 AnXQL 查询的重要建筑单元(building block),下面给出它的定义:

定义 3(路径表达式) (1){}表示一棵空树;(2){l=>T}表示一棵树,它的根有一条标记为 l 的输出边,下面附着一棵子树 T;其中 T 可以表示叶节点,此时 T 等于某文字值或空元素。由于{l=>T}只表示单一一棵树,可省略括号,简写为 l=>T;(3)$\{l_1=>T_1, l_2=>T_2,\cdots,l_n=>T_n\}$ 也表示一棵树,它有 n 个由同一节点发出的、分别标记为 $l_1,l_2,\cdots,l_n$ 的输出边,每个输出边分别附着子树 $T_1,T_2,\cdots,T_n$;(4)无论是边标签 l,还是子树 T_i(包括叶节点),都可以用变量表示;(5)2,3 或 2,3 的组合称为路径表达式。

AnXQL 查询语句由两个重要的部分组成,一个称为构造部分(construction part),另一个称为抽取部分(extraction part)。抽取部分由模式匹配条件表达式和称为过滤器(filter)的谓词条件表达式组成。构造部分由 select 子句来表达,而抽取部分由 where 子句来表达。所谓模式匹配就是用路径表达式来定义查询条件,将路径表达式作为一个模式去原文档中进行匹配,满足条件的 XML 文档片断被返回;而谓词表达式是指结果为布尔类型的条件表达式。

例如,针对与图 4 XML DTD[11]一致的 XML 文档,可能有查询请求 Q1 如下:

```
Q1:select RESULT=>{BOOK=>{TITLE
   =>$x,PRICE=>$y,author=>$z}}
where COMPUTISTIS=>BOOK=>{TITLE=>$x,PRICE=
      >$y,AUTHOR=>$z}}
      and $y(=50 in "www.a.b.c/bib.xml";
```

其中,where 子句由一个路径表达式和一个谓词表达式($y<=50)组成。如果进一步细分,查询 Q1 实际上由三部分组成:构造部分、匹配部分和过滤器部分。三个部分之间,数据的传递可以视为元组(tuple)的传递,元组具有扁平和顺序无关的特性,这是 AnXQL 查询语言的一个重要特征。"in"子句用来指定查询的文档所在的 URI(Unified Resource Identifier)。为简单起见,在下面的叙述中对单一数据查询的 AnXQL 语句省略 in 子句,实际上包装器所接收到的都是这样的 AnXQL 查询,应为查询处理器已将用户的查询请求分解为直接针对数据源的查询。首先,路径表达式到原文档中去匹配,结果是满足匹配模式的值组成元组集合($x, $y, $z),然后元组集合被传递给过滤器,变量 $y 小于等于50所对应的元组被保留,并被传递给构造部分按 select 子句的路径表达式为每一组元组加标签生成新的 XML 模型表示。

2. 关系型数据源包装器开发所面临的挑战

关系型数据源包装器的生成面临许多问题和挑战,总结如下,后面针对这些问题,将设计相应的技术和方法。

①关系型数据库系统中的数据是按表结构(tabular)来组织的,结构扁平(flat),并且是规范化的(normalized),而 XML 文档的结构却是带有自描述标记的、层次化的、嵌套的树型结构,并且是非规范化的(unnormalized)。如何解决关系模式两层结构(table 和 attribute)特性与 XML DTD 模式任意嵌套结构特性间相冲突的问题,如何合理地设计它们之间的映射关系,是我们需要处理的一个重要问题;

②由于多方面原因,数据源 XML DTD 与关系型数据库的 schema 可能采用完全不同的命名规则。这些原因包括,关系型数据库的 schema 定义不符合集成系统对数据源模式或全局模式定义的要求;或者,我们可能被要求在一个公共的 DTD 定义上建立数据集成系统等。由于 AnXQL 查询是建立在数据源的 DTD 上的,而 SQL 查询只对关系数据库的 schema 有效,反过来,SQL 查询的结果是在关系型 schema 的基础之上,而该结果必须转换成符合数据源 DTD 的 XML 文档,两个方向的处理都需要找到一个有效的映射机制。

③AnXQL 查询是针对层次结构的 XML 数据建立的查询语言,它的最大特点是带有模式匹配,或者说结构遍历(navigation)功能的路径表达式,如何将带有路径表达式(包括正则路径表达式)的 AnXQL 查询转换成相应的关系数据源能够理解的 SQL 查询是我们需要处理的一个关键问题;

④AnXQL 与 SQL 具有不同的语义,而且这种不同有时是难以调和的。如 AnXQL 中的分组运算与 SQL 中的分组运算(GROUP BY 子句),具有不同的语义,带有分组运算等不同语义运算符的 AnXQL 查询如何转换成 SQL 查询,查询结果如何处理等,也是我们需要解决的问题之一;

⑤外键(foreign key)是关系数据库系统进行数据组织的重要方式,然而在对应的 XML 数据表示中如何体现和包含这种数据组织方式也是我们必须考虑的问题。

3. 关系数据的 XML 描述

关系数据模型具有"表"结构表示,扁平和规范化的特点。相反,XML 数据却具有任意层次的嵌套结构和非规范化的特点,良好地表示二者数据之间的映射关系,是查询翻译和数据格式转换的起点。XML 语言本身的灵活性为我们找到问题的答案提供了可能。

利用 XML 技术,我们设计了称为 RTX(Relational To XML)的标记语言来表达关系数据库中的数据与 XML 数据之间的相互映射关系,如图2给出了这种标记语言的 DTD。

图3是利用 RTX 为图4的关系模式和图5的 XML DTD 建立的 RTX 映射描述。下面结合图3(为了说明问题方便,加了行号)的例子对图2的 DTD 中各元素和各属性(为了区别 XML 中的属性和关系模式中的属性,我们称关系模式中的属性为表属性),即 RTX 标记语言的含义做简单的说明:

·RDBTOXML 是映射表示的顶层节点，其属性 CONNECT 的值告诉包装器软件如何访问关系型数据库数据源。例如我们令

CONNECT="password/login@machine:portnum:SCIENTISTS"

则意味着，包装器以帐号 login，密码 password，在主机 machine 的 portnum 端口访问数据库 SCIENTISTS。

·MAPPING 元素可以有一个或多个(+号表示)，是表达关系数据与 XML 数据映射的基本单元。图3包含两个 MAPPING 单元，分别是第3行到第24行和第25行到第46行。它的子元素 XMLELEMENT 包含的字符串用以说明 XML 文档中某元素的名称；DBTABLE 包含的字符串用以说明上面的 XMLELEMENT 值所声明的 XML 元素对应的关系表属性所在的表的名称。如第4、5行表示 XML 元素 COMPUTIST 与关系表 computist 中的表属性的定义，具体的对应关系由 MATCH 来声明。

·MATCH 元素用以表达 XMLELEMENT 所对应的一个或多个属性。它包含三个子元素 SUBELEMENT，DBCOLUMN 和 FOREIGN；SUBELEMENT 是可选的元素(?号表示)，FOREIGN 可以有零个或多个(*号表示)。当 XMLELEMENT 只对应一个表属性的时候，则不需要 SUBELEMENT 元素，此时 XMLELEMENT 中的名称与 DBCOLUMN 中的表属性名直接对应；当 XMLELEMENT 对应多个表属性的时候，说明 XMLELEMENT 中的 XML 元素还包含其它子元素，SUBELELMENT 和 DBCOLUMN 来表达 XML 元素与表属性的一一对应关系，图3的例子就是这种情况。

·FOREIGN 是一个非常重要的概念，它将两个或两个以上的数据库表通过外键联系起来。图3的第15行到第22行表示 XML 元素 ORGANIZATION 所包含的值与表 organization 中表属性 org_name 相对应，而这种对应关系是由表 computist 的外键属性 org_id 和 organization 中的主键属性 id 间建立连结来实施的，即第16行和第17行 DBCOLUMN 中的 org_id 与 FOREIGN 的属性 KEY 的 id 表示同一概念(请参照图5的关系模式)。FOREIGN 的设计使得扁平的数据库表能够表示嵌套关系。

·TOPLEVEL 为所定义的一个或多个 MAPPING 单元加上一个顶层元素，这是 XML 语法结构的要求。

图3的 RTX 描述在图5的关系模式和图4 XML DTD 间建立了映射关系。图3的例子虽然只包含两个 MAPPING 元素，但充分说明了 RTX 语言表达映射关系的方式和能力，如果实际情况需要，可以添加更多的 MAPPING 元素，以定义更复杂的关系模式与 XML DTD 之间的映射关系。实际上，RTX 语言足以表达任意复杂的关系模式与 XML 模式之间的映射，但此时，我们可能需要做一些必要的扩充，如可以像〈TOPLEVEL〉那样，在RTX定义中增加〈SECONDLEVEL〉，甚至〈THIRDLEVEL〉标签，为不同语义的 MAPPING 进行分组，形成更深的嵌套结构，为了说明问题方便，本文的 RTX 并没用作这样的定义。而且 RTX 映射定义的表达具有双向性，即利用这种映射表达不但能够将关系数据表示成 XML 数据，而且可以用于将 XML 数据存储为关系数据表示，虽然后者并不是本文所要讨论的重点。

开发 RTX 语言来表示映射关系只是解决问题的开始。由于 RTX 在表示映射关系时书写较复杂，特别是当要映射的数据库表属性较多，存在多个数据库表通过外键发生嵌套关系的时候更是如此。因此有必要开发图形界面工具来辅助集成系统的用户或开发工程师来定义 RTX 映射关系。配合本论文的研究，我们设计了称为 Mapper 的工具来辅助生成关系数据到 XML 数据的 RTX 映射。篇幅所限，此处没有给出 Mapper 的说明。

```
<!ELEMENT RDBTOXML(TOPLEVEL,MAPPING+)>
<!ATTLIST RDBTOXML CONNECT CDATA #REQUIRED>
<!ELEMENT MAPPING(XMLELEMENT,DBTABLE,MATCH+)>
<!ELEMENT MATCH(SUBELEMENT,DBCOLUMN,FOREIGN*)>
<!ELEMENT FOREIGN(DBTABLE, MATCH+)>
<!ATTLIST FOREIGN KEY CDATA #REQUIRED>
<!ELEMENT XMLELEMENT (#PCDATA)>
<!ELEMENT DBTABLE (#PCDATA)>
<!ELEMENT DBCOLUMN (#PCDATA)>
```

图2 RXT 标记语言的文档类型定义

```
1<RDBTOXML CONNECT="db connect string">
2 <TOPLEVEL>COMPUTISTS</TOPLEVEL>
3 <MAPPING>
4   <XMLELEMENT>COMPUTIST</XMLELEMENT>
5   <DBTABLE>computist</DBTABLE>
6   <MATCH>
7       <SUBELEMENT>COMTITLE</SUBELEMENT>
8       <DBCOLUMN>title</DBCOLUMN>
9   </MATCH>
10  <MATCH>
11      <SUBELEMENT>NAME</SUBELEMENT>
12      <DBCOLUMN>name</DBCOLUMN>
13  </MATCH>
14  <MATCH>
15      <SUBELEMENT>ORGANIZATION</SUBELEMENT>
16      <DBCOLUMN>org_id</DBCOLUMN>
17      <FOREIGN KEY="id">
18          <DBTABLE>organization</DBTABLE>
19          <MATCH>
20              <DBCOLUMN>org_name</DBCOLUMN>
21          </MATCH>
22      </FOREIGN>
23  </MATCH>
24 </MAPPING>
25 <MAPPING>
26   <XMLELEMENT>BOOK</XMLELEMENT>
27   <DBTABLE>book</DBTABLE>
28   <MATCH>
29      <SUBELEMENT>TITLE</SUBELEMENT>
30      <DBCOLUMN>btitle</DBCOLUMN>
31   </MATCH>
32   <MATCH>
33      <SUBELEMENT>PRICE</SUBELEMENT>
34      <DBCOLUMN>bprice</DBCOLUMN>
35   </MATCH>
36   <MATCH>
37      <SUBELEMENT>AUTHOR</SUBELEMENT>
38      <DBCOLUMN>au_id</DBCOLUMN>
39      <FOREIGN KEY="id">
40          <DBTABLE>computist</DBTABLE>
41          <MATCH>
42              <DBCOLUMN>name</DBCOLUMN>
43          </MATCH>
44      </FOREIGN>
45   </MATCH>
46  </MAPPING>
47</RDBTOXML>
```

图3 RXT 语言描述的关系数据与 XML 数据的映射

```
<!ELEMENT COMPUTISTS(COMPUTIST+,BOOK*)>
<!ELEMENT COMPUTIST(COMTITLE,NAME,
        ORGANIZATION)>
<!ELEMENT BOOK (TITLE, PRICE, AUTHOR)>
<!ELEMENT COMTITLE (#PCDATA)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT ORGANIZATION (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT AUTHOR (#PCDATA)>
```

图 4 与关系数据源对应的 XML DTD

```
computist(id, title, name, org_id);
organization(id, org_name, org_address);
book(id,btitle,bprice,au_id);
```

图 5 一个简单的关系数据模式

4. AnXQL 查询翻译与数据格式转换

AnXQL 查询与 SQL 查询针对不同的数据模型而构造，它们具有完全不同的语义、表达能力和语法结构，而且关系模式和 XML 模式，即使它们定义或描述的是客观世界中的同一系列实体，却可能采用完全不同的命名规则。我们只有依靠 RTX 映射来帮助实施 AnXQL 查询到 SQL 查询的翻译，或关系数据格式到 XML 数据格式的转换。本小节将讨论如何将 AnXQL 查询翻译成 SQL 查询，以及如何将关系数据转换为 XML 数据格式。

为了表达问题方便，我们先给出“规范型路径表达式”的概念。

定义 4（规范型路径表达式 NFPE，Normal Form Path Expression） 是指不含有正则表达式(regular expression)的路径表达式，且该路径表达式为一个不带有分支的线性结构。所有的路径表达式都是 NFPE 的 AnXQL 查询称为规范型的 AnXQL 查询，简称 NFA(Normal Formal AnXQL)。

下面我们分若干种情况说明如何将一个 AnXQL 查询转换成 SQL 查询，同时说明每一种情况中，如何将 SQL 查询返回的元组加上标签(tagging)生成 XML 数据格式。我们按以下几种情况论述：简单路径表达式的 AnXQL 查询，带有连结运算(join)的 AnXQL 查询，带有正则路径表达式的 AnXQL 查询和带有标签变量的 AnXQL 查询，以及两种语言语义冲突情况下的 AnXQL 查询如何转换成 SQL 查询。然后总结出 AnXQL 查询到 SQL 查询的通用算法。

本小节中所举的查询转换的例子在上小节的例子上构造。

(1)*带有简单路径表达式的 AnXQL 查询* AnXQL 查询在图 4 的 DTD 上构造，它们被转换为在图 5 关系模式上的 SQL 查询。

例1 设有 AnXQL 查询 Q2如下，它查询“John Smith”所参与写作的图书。

```
Q2:
Select RESULT=>{TITLE=>$x}
Where COMPUTISTS=>BOOK=>{TITLE=>$x, AUTHOR
   =>"John Smith"};
```

首先将查询 Q2转换为规范型的 AnXQL 查询 NFA，则原来的 where 子句被转换成等价的两个 NFPE 表达式，如下：

```
COMPUTISTS=>BOOK=>TITLE=>$x
COMPUTISTS=>BOOK=>AUTHOR=>"John Smith"
```

扫描图3的 RTX 映射，为每一个路径表达式填写如表1的数据结构，为了叙述方便，我们称该数据结构为 DST(Data Structure for Transformation)。

表1 为查询 Q2翻译填写的数据结构

PE	对应表	对应属性	外键关系	对应变量或常量
1	book	btitle	nil	$x
2	computist	name	computist.id=book.au_id	"John Smith"

表1的 DST 数据结构填写完毕以后，根据该数据结构生成针对图5关系模式的 SQL 查询，过程如下：将“对应表”中的数据库表加入到 SQL 查询的 from 子句中，并用“,”号分隔不同的项目；如果“对应变量或常量”项目中是变量，将与该变量对应的“对应表”与“对应属性”中的项目用“.”号连结，加入到 SQL 查询的 select 子句中，不同的项目之间用“,”号分隔；“外键关系”中的表达式加入到 SQL 查询的 where 子句中；如果“对应变量或常量”项目中是常量，将与该常量对应的“对应表”与“对应属性”中的项目用“.”连结，与该常量生成等式加入到 SQL 查询的 where 子句中；由于 SQL 查询 Q2对应的是一个路径表达式，所以该 where 子句中的两个表达式用 and 连结，从而有与 Q2对应的 SQL 查询，如下(Q3)：

```
Q3:
SELECT computist.title
FROM book, computist
WHERE computist.id = book.au_id and computist.name = "John
      Smith";
```

加标签运算是一个很直接的过程，只要系统在 SQL 查询返回的变量 $x 的元组流上，按 AnXQL 的构造语句(select 子句)的要求加上标签，生成 XML 文档即可。

(2)*带有连结运算的 AnXQL 查询* 我们再举一个较复杂的例子如下：

例2 设有查询 Q4，它要求列出任意一本图书的作者和他所在的机构。从而 AnXQL 查询如下，它在图4的 XML DTD 上构造：

```
Q4:
Select RESULT=>{BOOK=>{TITLE=>$x,
   AUTHOR=>$y,ORGANIZATION=>$z}}
Where COMPUTISTS=>{COMPUTIST=>{NAME=>$y,OR-
      GANIZATION=>$z}} and
      COMPUTISTS=>{BOOK=>{TITLE=>$x,AUTHOR
      =>$y}};
```

显然，这是一个带有连结运算的 AnXQL 查询，BOOK 元素的 AUTHOR 与 COMPUTIST 元素的 NAME 进行连结，此处假设没有作者重名的情况出现。首先生成查询 Q4的 NFA 形式，得到与 where 子句对应的四个 NFPE 路径表达式如下：

```
COMPUTISTS=>COMPUTIST=>NAME=>$y
COMPUTISTS=>COMPUTIST=>ORGANIZATION=>$z
COMPUTISTS=>BOOK=>TITLE=>$x
COMPUTISTS=>BOOK=>AUTHOR=>$y
```

表2 为查询 Q4翻译填写的数据结构

PE	对应表	对应属性	外键关系	对应的变量或常量
1	computist	Name	Null	$y
2	organization	Org_name	Computist.org_id=organization.id	$z
3	Book	Btitile	Null	$x
4	Computist	Name	Book.au_id=computist.id	$y

扫描图3的 RTX 映射描述，为每一个 NFPE 路径表达填写如表2的 DST 数据结构。

访问表2已填写好的数据结构，将“对应表”中的数据库表加入 SQL 的 from 子句中（去掉重复项）；“对应的变量或常量”中都是变量，所以将“对应表”与“对应属性”中的各对应项用“.”号连结加入 select 子句中，并用“,”号分隔（去掉重复项）；将“外键关系”中的两个等式加入到 where 子句中，由于原 AnXQL 查询中无“or”条件，因此两个等式仍然用“and”连结，从而得到在图5关系模式上的 SQL 查询如下（Q5）：

```
Q5:
SELECT book. btitle,computist. name,organization. org_name
FROM computist, book, organization
WHERE computist. id=book. au_id And
      organization. id=computist. org_id;
```

最后在返回的（$x，$y，$z）元组上按构造子句的要求加标签。

还要说明的一点是，以上我们所举的 AnXQL 查询的例子其 where 子句中都是路径表达式，没有一般的谓词表达式，例如在查询 Q4中只要求书价大于50元的图书的信息，则应在其 where 子句中增加如下的路径表达式和谓词表达式：

```
COMPUTISTS=>BOOK=>PRICE=>$w
$w>50
```

这时只要将 DST 数据结构中与变量 $w 对应的“对应表”和“对应属性”中的项目用“.”号连结，替换谓词表达式 $w>50 中的变量 $w，将其加入到 SQL 的 WHERE 子句中，并对路径表达式做适当的处理即可。

(3)带有正则路径表达式和标签变量的 AnXQL 查询

带有正则路径表达式的查询是 XML 数据模型上的一种非常重要的查询语义表达。那么如何将带有正则路径表达式的 AnXQL 查询翻译成 SQL 查询呢？为此，首先引入了路径推断（path inference）的概念。所谓路径推断，就是对 AnXQL 查询中的正则路径表达式根据所查询文档的 DTD 进行特化（specialized）处理，将 AnXQL 查询中出现的正则路径表达式根据其所要查询的 XML 数据的 DTD 重写为不含正则路径表达式的等价的一个或多个简单的路径表达式，然后使用上面翻译 Q2和 Q4类似的方法将推断处理后的 AnXQL 查询转换为 SQL 查询。经过推断处理的查询转换与上面的方法相同。路径推断是一个重要的手段，不但用于正则路径表达式，而且可以用于带有标签变量的路径表达式，我们举例如下，设有在图4 XML DTD 上的 AnXQL 查询（Q6）：

```
Q6:
Select RESULT=>{_=>$y}
Where COMPUTIST=>{_=>$y};
```

显然，这是一个带有正则路径表达式的 AnXQL 查询，因为 where 子句中的路径表达式含有通配符“_”。首先对 Q6进行路径推断，得到三个 AnXQL 查询，分别如下：

```
select RESULT=>{COMTITLE=>$y}
where COMPUTIST=>{COMITITLE=>$y};
select RESULT=>{NAME=>$y}
where COMPUTIST=>{NAME=>$y};
select RESULT=>{ORGANIZATION=>$y}
where COMPUTIST=>{ORGANIZATION=>$y};
```

对以上的 AnXQL 查询分别进行翻译转换，然后对三个 SQL 查询返回的结果分别进行加标签处理，将最后的结果统一到一个 XML 元素下即可。

(4)带有分组运算的 AnXQL 查询 分组（grouping）运算，无论是对 XML 数据模式上的 AnXQL 查询，还是对关系模式上的 SQL 查询来说都是一个非常重要的运算，然而由于 XML 数据模型与关系模型截然不同，使得 AnXQL 的分组运算与 SQL 的分组运算具有完全不同的语义。设有图4 XML DTD 模式上的查询 Q10，它要求按照作者名将她/他所参与写作的图书进行分组，如下：

```
Q10:
Select RESULT=>{[$y] AUTHORnBOOK=>
       {AUTHOR=>$y,[$x] TITLE=>$x}}
Where COMPUTISTS=>{BOOK=>{TITLE=>
       $x, AUTHOR=>$y}};
```

元组集合（$x，$y）被传递给构造部分，[$y]表示为元组集合中的每一个不重复的绑定到 $y 变量的值生成一个树结构 AUTHORnBOOK=>{AUTHOR=>$y,[$x] TITLE=>$x}，而树结构中的[$x]表示为与这样的 $y 值对应的每一个 $x 值生成一个树结构 TITLE=>$x。[$y]和[$x]是 AnXQL 表达分组运算的方式之一[8]。

显然 SQL 中的分组运算无法表达像 Q10这样的查询语义。那么像这样的 AnXQL 查询请求如何翻译成对应的 SQL 查询呢？分析 SQL 语义，只有按作者进行排序（ORDER BY），才能将同一作者与他所写作的图书放在一起，这样就可以在由关系数据流生成 XML 数据文档的时候，对同一作者加一个 AUTHOR 标签（tag），与该作者对应的多本图书加每一个加 TITLE 标签，形成 AnXQL 查询分组的语义。结合上面的技术，可以得到 SQL 查询 Q11如下：

```
Q11:
SELECT computist. name, book. btitle
FROM book,computist
WHERE computist. id=book. au_id
ORDDER BY computist. name;
```

总结以上的观察，很容易得到 AnXQL 到 SQL 查询的转换算法，我们把该算法命名为 ATS（AnXQL To SQL），用自然语言的形式叙述如下：

算法：ATS

```
Input：一个 AnXQL 查询，一个 XDRD 映射描述
Output：一个或多个 SQL 查询，AnXQL 查询的构造语句（select 子句）
Switch（AnXQL 查询的类型）
case 带有正则路径表达式或标签变量的或有独立的根路径表达式的：
   进行路径推断，得到等价的简单路径表达式的 AnXQL 查询；
   将简单路径表达式的 AnXQL 查询转换为 NFA；
   将结果 NFA 做为参数调用函数 CreateSQL(NFA)；
   break；
case 带有简单路径表达式的：
   if AnXQL 查询不是 NFA
   then 转换为 NFA；
   调用函数 CreateSQL(NFA)；
   break；
End Switch
函数：CreateSQL(NFA)
分解 NFA 的构造语句和条件语句；
对条件语句中的每一个路径表达式，扫描 RTX 映射，填写数据结构 DST；
对填写好的 DST 结构作如下操作：
step 1：将“对应表”中的数据库表加入到 SQL 查询的 from 子句中，用逗号分隔；
step 2：if “对应变量或常量”为变量
          与该变量对应的“对应表”与“对应属性”中的项目用“.”号连结，加入到 SQL 查询的 select 子句中，不同的项之间用“,”号分隔；
        else if “对应变量或常量”为常量
          与该常量对应的“对应表”与“对应属性”中的项目用“.”连结，与该常量生成等式加入到 SQL 查询的 where 子句中；
step 3：“外键关系”中的表达式加入到 SQL 查询的 where 子句中；
step 4：if NFA 有谓词表达式
          将谓词表达式中的变量用 DST 中对应的“对应表”项目加“.”加“对应属性”项目替换，其它不变，加入 SQL 查询的 where 子句中；
step 5：if NFA 有分组运算
          将分组运算转换为 ORDER BY 运算，分组变量转换为与该变量对应的“对应表”项目加“.”加“对应属性”项目，加入 SQL 查询的 where 子句；
step 6：SQL 查询中各谓词表达式用适当的“and”或“or”连结；
Return NFA 的构造语句，SQL 查询；
```

5. 关系型数据源包装器系统的模块组成

根据以上各小节的分析，我们可以得出一个结论，面向关

系数据库系统数据源的包装器最起码应该具有两个功能模块，一个负责将 AnXQL 查询转换为 SQL 查询，另一个负责将 SQL 查询返回的元组流加相应的标签转换为 XML 数据格式。我们将前者称为 Translator，将后者称为 Generator。图6是一个关系型数据源包装器功能模块和运行流程示意图。图中的 Translator 主要以上节给出的 ATS 算法为主构造，它以 AnXQL 查询和 RTX 映射为输入，将 AnXQL 查询的构造部分(select part of AnXQL)传递给模块 Generator，将 SQL 查询传递给关系型数据库系统(RDB)；Generator 模块以关系型数据库系统返回的元组流(tuple stream)和 AnXQL 查询的构造部分为输入生成 XML 数据。

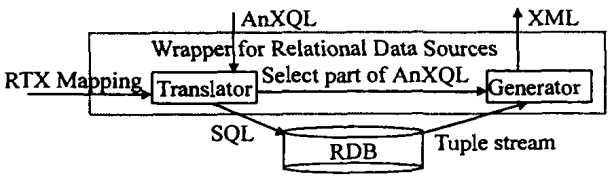

图6 关系型数据源打包器功能模块示意图

6. 相关研究工作

XML 语言正在成为互联网上数据交换的标准，而关系数据库系统是使用最广泛的数据存储和管理工具。因此，如何将关系数据格式转换为 XML 数据格式，或相反，如何用关系数据库系统来存储 XML 数据，近来成为研究界和产业界一个非常热门的课题[12~14]。

文[14]的研究目的与本部分的研究目的类似，但是采用完全不同的方法。它设计了一种称为 SilkRoute 的重型系统来定义关系数据的 XML 视图和将关系数据库的查询结果转换为 XML 数据格式。为了实现这个目的，SilkRoute 将关系查询与 XML 查询语言相结合，开发了另一种称为 RXL 的查询语言，用于定义虚拟的 XML 视图，系统的用户查询以 XML-QL 查询语言表达。当系统接收到用户查询的时候，它将 XML-QL 与 RXL 定义的视图进行复合(composition)，生成针对关系数据源的 RXL 查询。因为 SilkRoute 系统设计目标是一个独立的系统，所以所采用的方法和处理问题的范畴相对复杂。同时，SilkRoute 要求设计全功能的查询处理系统。我们的包装器，是作为集成系统一部分的一个轻型包装器系统，它可以充分利用关系型 DBMS 的查询能力，RTX 语言和映射的设计，使得一个复杂的问题得到了很好的解决。

Oracle8i 数据库系统[12]的方法和技术，主要是基于对 SQL 查询语言的扩展，这就需要设计新的查询执行引擎。而且 Oracle8i 是在"对象关系数据库"的概念基础上设计的新型数据库系统，它的对象特性与 XML 数据的组织方式一致，所以 Oracle8i 不但可以将查询转换为 XML 数据，而且可以直接存储 XML 数据。然而，它的技术并不适用于我们的问题情境。

其它还有一些研究原型和商业系统，在此我们不做一一对比，下面给出几点，说明本文所采用的方法和解决问题的范畴与其它系统的不同。

我们只研究关系型数据到 XML 数据的单向转换，不考虑从 XML 到关系型数据的转换，这使问题相对简单，也使我们的系统轻型化。

我们只考虑一个关系型数据源只针对一个 XML 数据文档，或者说一个 XML DTD 的情况，当关系数据库较复杂，不便于用一种 DTD 来表达的时候，可以用两个或两个以上的 DTD 来表达，生成多个 RTX 映射，不同的 DTD 和对应的 RTX 视为不同的数据源加入集成系统。

因为在查询处理的环节中已经对用户查询进行了处理，关系型数据源接收到查询必然只针对一个 XML 文档进行操作，所以不存在多个文档间的连结运算，只存在一个文档的不同域的连结运算。

我们的研究目的与其它项目最大的不同在于，我们只关心如何将关系型数据转换成符合某个 DTD 的 XML 文档，而不关心它的反向操作。我们的目的是设计一个轻型(lightweight)的关系型数据源包装器软件构造的技术和工具，从而为整个数据集成系统服务。

结束语 本文给出了一种数据集成系统中关系数据源包装器设计与实现的轻型方法，很好地解决了 XML 查询到 SQL 查询的翻译问题，从而使得扁平的、规范化的关系数据到 XML 数据的转换成为可能。

参 考 文 献

1 Bray T, Paoli, J, Sperberg-McQueen C. Extensible Markup Language(XML) 1.0. Http://www.w3.org/xml/1998/06/xmlspec-report-19980910.htm

2 Florescu D, Levy A, Mendelzon A. Database techniques for the World-Wide Web: A survey. SIGMOD Record, 1998, 27(3): 59~74

3 Chawathe S, et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In: Proc. of IPSJ Conf. 1994. 7~18

4 Pottinger R, Levy A. A Scalable Algorithm for Answering Queries Using Views. VLDB 2000, Cairo, Egypt

5 Kushmerick N, Doorenbos R, Weld D. Wrapper induction for information extraction. In: Proc. of the 15th Intl. Joint Conf. on Artificial Intelligence, 1997

6 Christophides V, Cluet S, Sim'eon J. On wrapping query languages and efficient XML integration. In: Proc. of ACM SIGMOD Conf. on Management of Data, Dallas, Texas, May 2000

7 Levy A Y, Rajaraman A, Ordille J J. Querying Heterogeneous Information Sources using Source Descriptions. In: Proc. 22nd VLDB Conf. Bombay, India, 1996

8 李效东. 自治异构数据源的集成查询处理:[中科院软件所博士论文](手稿)

9 李效东. XML 查询的代数表示及其查询优化.(待发表)

10 Buneman P, et al. A query language and optimization techniques for unstructured data. In: Proceedings of ACMSIGMOD International Conference on Management of Data, Montreal, Canada, June 1996. 505~516

11 Bosak J, et al. W3C XML Specification. DTD. http://www.w3.org/XML/1998/06/xmlspec-report.thm

12 Banerjee S, et al. Oracle8i--The XML Enabled Data Management System. In: Proc. of the 16th Intl. Conf. on Data Engineering, San Diego, California, 2000

13 IBM DB2 XML Extender. http://www-4.ibm.com/software/data/db2/extenders/xmlext/index.html

14 Fernandez M, Suciu D, Tan W. SilkRoute: trading between relations and XML. In: Proc. of the WWW9, Amsterdam, 2000