文章编号:1001－9081(2004)08－0069－03

一种可变数据结构存储处理模型

朱庆生,葛 亮
(重庆大学 计算机学院,重庆 400044)
(qszhu@cqu.edu.cn)

摘 要:针对电子商务中对可变数据结构存储和处理的应用要求,提出了一种通用的数据存储模型及其处理框架。其基本思想是在关系数据存储模型的基础上引入了可变实体存储模型,在模型的描述和处理中运用了 XML、XQuery、DOM 等技术。示例程序证明了其可行性。

关键词:可变数据结构;数据模型;弹性存储
中图分类号:TP311;TP312 文献标识码:A

Storing and processing model for changeable data structure

ZHU Qing-sheng, GE Liang
(*College of Computer Science, Chongqing University, Chongqing* 400044, *China*)

Abstract: This paper introduced a general data storing and processing model, which fit to the changeable data structure in E-business. The main idea was to set up a changeable entity storing model based on the relational data storing model. In the model, description and process involved some technologies such as XML, XQuery and DOM. The model proved to be feasible by a deamon program.

Key words: changeable data structure; data model; flexible storing

从 1996 年 IBM 提出“电子商务”的概念开始至今,电子商务得到了迅猛的发展。从最初的企业上网建立主页发布一些静态信息,到后来企业将内部的信息系统和业务流程整合在一起以实现真正的在线交易,人们从中得到了以前传统商务所不及的方便和快捷。但同时也出现了一些问题,例如:对于网络营销中的客户信息管理,由于不同企业对客户信息的侧重点不同,顾客信息成为一种可变的数据结构,而传统关系数据模型很难对可变数据结构进行统一建模,由此系统的可伸缩性大大降低,使得开发人员往往为满足用户不断变化的需求而频繁修改系统。如何实现具有灵活可变特点的随需应变电子商务[1]成为当前电子商务研究的重心。

电子商务的核心功能之一就是存储、处理和交换各种信息。为了统一存储和处理电子商务中各种不同的数据结构,特别是可以动态变化的可变数据结构,本文提出了一种关系数据库与 XML[2] 相结合的混合模型,称为可变实体存储模型。

1 可变实体存储模型

在电子商务应用中有很多随用户需求和时间变化而变化的数据结构,例如顾客信息、商品信息条目、商家信息条目等。为便于问题的讨论,我们将可变数据结构定义为可变实体。

定义 1 对于某个实体,如果其属性集可以在系统运行时动态改变,则该实体为可变实体。在每一时刻都具有相同属性集定义的一类可变实体,称为可变实体集。

为简化问题和保证存储数据的有效性,我们作以下的约定:

1) 主要考虑实体属性集(即不包括实体的主码和外码)的动态改变,不考虑实体间联系(外码集)的动态改变。

2) 可变实体的属性集虽然可以随需求和时间动态改变,但某一时刻只能有且只有一个确定的属性集,以表示当前可变实体的有效属性。

3) 为了保证数据的一致性,由应用决定是否允许在可变实体集的属性集定义中增加一个新属性,而该新属性不能取空值。如果允许,则必须要同时为实体集中已有的实体增加相应的属性,并赋一默认值。

4) 在可变实体属性集的 XML 表示中,当没有某个属性标签时,则表示该标签所表示的属性值为空(null)。

当前,绝大多数电子商务的应用都使用关系数据库来存储和管理数据。实体在关系数据库中的表示是:一张二维表表示一个实体集,其中表的每一行表示一个实体,表的每一列表示实体的一个属性(如图 1(a)所示)。为了反映实体的具体属性,这张二维表的结构在系统设计时就要有明确的定义,比如二维表有哪些列、每列代表什么含义等。这种存储模型很难在系统运行时动态改变表结构,因此不便存储可变实体。

XML 是一种结构化的语言,其优点之一就是通过标记可以实现很好的易变性,这正好有利于解决传统关系数据存储模型不便存储可变实体的问题。通过将 XML 融入到关系数据库中,就得到了可变实体存储模型。该模型的主要思路是:用 XML 来表示实体的属性集,利用其易变性来反映实体属性的变化,然后将 XML 存放在关系数据库相应实体的二维表的一个固定列中。这样,当实体属性发生改变时,只需要改变 XML 的内容,而不需要改变二维表的表结构,从而方便地实现了可变实体的存储。该模型的结构如图 1(b)所示。

从图 1(b)可看出,在该模型中,一个可变实体集仍然由

收稿日期:2004－02－19

作者简介:朱庆生(1956－),男,安徽人,教授,博士生导师,主要研究方向:多媒体技术、软件工程;葛亮(1980－),男,重庆人,硕士研究生,主要研究方向:电子商务、网络信息系统、软件开发环境。

一张二维表表示,表的每一行表示一个可变实体。二维表的列由三部分组成: ID(可变实体的唯一标识,作为实体的主码)、外码集(包含零个或多个列,用于存放与该实体集相关的其他实体的外码)和属性集(一个固定列,用于存放可变实体属性集的 XML 文档)。

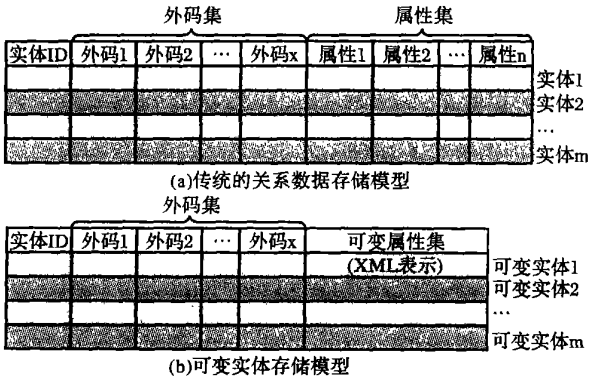

图 1 传统关系数据存储模型和可变实体存储模型的比较

其中实体属性集的 XML 文档由< propertySet> 和< property> 两种元素构成。文档的根节点是< propertySet> 元素,它表示了实体的属性集;它的子节点是由零个、一个或多个< property> 元素组成的序列,每个< property> 元素表示实体的一个属性,其中< property> 元素有三个属性,“name”、“type”和“nullable”,分别表示属性的名称、取值类型和是否允许为空;而< property> 元素的值就是实体属性的值。这里要求实体属性值只能是简单数据类型,如布尔值、整数、实数、字符串、日期等。

用 XML Schema[3] 进行规范化定义,其内容如下:

```
<xs:element name="propertySet">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="property" nillable="true"
        minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:anySimpleType">
              <xs:attribute name="name" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:string"/>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="type" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="xs:
                      boolean"/>
                    <xs:enumeration value="xs:
                      number"/>
                    <xs:enumeration value="xs:
                      string"/>
                    <xs:enumeration value="xs:
                      date"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="nullable" type="xs:
                boolean" use="optional" default="true"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

2 可变实体处理框架

由于在可变实体存储模型中,实体的属性存放在 XML 文件中,因此不能直接用 SQL 来操纵实体数据。为此,我们引入可变实体处理框架来完成可变实体的存储、访问和修改操作。可变实体处理框架由 4 个部分组成,其结构如图 2 所示。

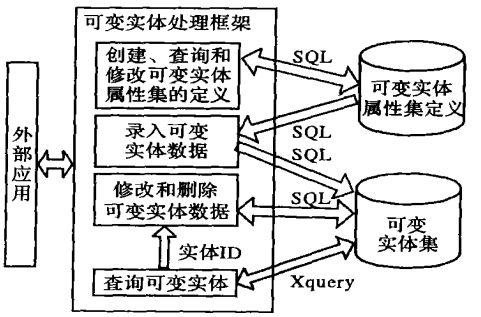

图 2 可变实体处理框架的结构

2.1 可变实体属性集的定义、查询和修改

该部分根据属性集的 XML 格式定义,按照外部应用的要求,创建和修改某个可变实体的属性集定义,并将属性集定义存放在关系数据库中,当需要时,也可从关系数据库中读出属性集定义的内容。

2.2 可变实体数据的录入

该部分根据属性集的定义,与外部应用交互,创建相应的可变实体数据,并存入关系数据库中。存入操作用 SQL 中的 INSERT 语句。

2.3 可变实体数据的修改和删除

该部分用于对某个具体的可变实体进行修改和删除操作。定位要操作的可变实体,需要给出该可变实体的 ID。如果需通过可变实体的属性进行定位,则需要先通过查询得到可变实体的 ID,再进行操作。修改操作用 SQL 中的 UPDATE 语句完成,而删除操作用 SQL 中的 DELETE 语句完成。

2.4 可变实体数据的查询

该部分是根据输入的查询条件得到需要的可变实体信息,这是处理框架中最重要的部分。为了满足用户需要,可变实体处理框架对以往在关系数据库中进行查询操作的 SELECT 语句进行了扩充,加入了 XML 的查询语言 XQuery[4]。

为便于说明,我们对以下符号进行约定:“entity-table”代表某个可变实体集的二维表;“entity-table-n”代表第 *n* 个可变实体集或非可变实体集的二维表,当多表查询时使用;“id”代表可变实体的 ID 列的列名;“foreign-key-n”代表可变实体的第 *n* 个外码列的列名;“xxx”代表某个具体的数值,其数据类型根据上下文意思确定。

下面分三种情况介绍可变实体数据的查询和处理策略。

2.4.1 查询条件中不包含可变实体的属性

此时可以直接用 SELECT 语句根据实体 ID 进行查询。

SELECT< 目标列表达式> FROM entity-table WHERE id= xxx

或者

SELECT＜目标列表达式＞ FROM entity-table-1, entity-table-2, …, entity-table-n WHERE [查询条件]

2.4.2　查询条件中只包含可变实体的属性

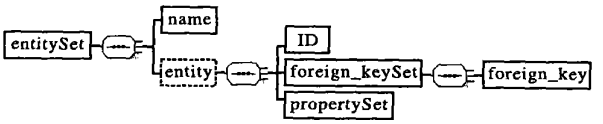

图 3　实体集的 XML 文档结构

此时，首先把待查询的可变实体集转变成一个 XML 文档，即把每个可变实体从关系数据库中取出来，将其合并成一个总的 XML 文档，从而得到实体集的 XML 文档。一个实体集对应一个 XML 文档。实体集的 XML 文档的结构如图 3 所示。实体集的 XML 文档结构具体描述为：

```
＜xs:element name= "entitySet"＞
  ＜xs:complexType＞
    ＜xs:sequence＞
      ＜xs:element name= "name" type= "xs:string"/＞
      ＜xs:element name= "entity" minOccurs= "0"
        maxOccurs= "unbounded"＞
        ＜xs:complexType＞
          ＜xs:sequence＞
            ＜xs:element name= "ID" type= "xs:string"/＞
            ＜xs:element ref= "foreign-keySet"
              minOccurs= "0"/＞
            ＜xs:element ref= "propertySet"/＞
          ＜/xs:sequence＞
        ＜/xs:complexType＞
      ＜/xs:element＞
    ＜/xs:sequence＞
  ＜/xs:complexType＞
＜/xs:element＞
＜xs:element name= "foreign-keySet"＞
  ＜xs:complexType＞
    ＜xs:sequence＞
      ＜xs:element name= "foreign-key" nillable= "false"
        minOccurs= "0" maxOccurs= "unbounded"＞
        ＜xs:complexType＞
          ＜xs:simpleContent＞
            ＜xs:extension base= "xs:string"＞
              ＜xs:attribute name= "name" type= "xs:string"
                use= "required"/＞
            ＜/xs:extension＞
          ＜/xs:simpleContent＞
        ＜/xs:complexType＞
      ＜/xs:element＞
    ＜/xs:sequence＞
  ＜/xs:complexType＞
＜/xs:element＞
```

其中＜entity＞元素的子元素＜propertySet＞引用了前面定义的＜propertySet＞元素。

然后，用 XQuery 对生成的 XML 文档进行查询，此时采用 XQuery 中的 FLWOR 表达式[4]，返回的结果集用 XML 表示。假定第 n 个实体集生成的 XML 文档的文件名用 dx-n.xml 表示，那么对于可变实体模型的查询，可使用以下几种表达式：

1) FOR $variable-name IN document("dx-n.xml")　//entity

是将一个变量与第 n 个实体集中的某个实体绑定，实现在源表中遍历实体的目的；

2) FOR $variable-name-1 IN document("dx-1.xml")　//entity
FOR $variable-name-2 IN document("dx-2.xml")
//entity[attribute-name = $variable-name-1
//foreign-key[@name = "foreign-key-name"]]

是将变量 variable-name-1 与第 1 个实体集中的某个实体绑定；将变量 variable-name-2 与第 2 个实体集中的某个实体绑定；并且其两个变量对应的实体用外码进行连接。实现在多表查询时，多张源表的连接操作。

3) WHERE＜条件表达式＞

通过设置条件表达式来限制返回的元组集，只有满足条件表达式为真的元组才被返回。

4) ORDERBY 子句

用于指定结果集中元组的排列顺序。如果实际的 XQuery 实现中没有提供该功能，也可由应用自行完成。

5) RETURN 子句

将查询得到的元组集合格式化成需要的形式并将其返回，以供外部应用使用。

2.4.3　查询条件中既包含可变实体 ID 又包含可变实体属性

此时采用两次查询的方式来完成，即先用 2.4.1 中所述的方法，对源实体集，根据可变实体 ID 的查询条件进行一次查询，得到一个中间结果；然后用 2.4.2 中所述的方法，对得到的中间结果，根据可变实体属性的查询条件进行一次查询，得到最终结果。

3　示例程序

3.1　示例程序的应用背景

商家在网站上出售商品，一般都希望顾客先输入个人信息，如姓名、性别、职位、受教育程度、收入水平等，以建立顾客信息档案；然后让系统记录顾客在网上的购物行为，如买了什么商品等；最后从顾客信息档案和顾客行为记录中查询数据，进行数据分析，得出一些有价值的信息，如月收入在3 000～5 000元的人喜欢买时尚商品等信息，从而指导商家更有针对性地采取营销策略。

从这个实例中可以看出，对顾客信息档案和顾客购物行为的有效存储和查询是实现上述功能的基础。而对于顾客信息档案的处理而言，不同的商家往往有不同的侧重点，甚至同一个商家在不同的时期也有不同的侧重点，从而导致顾客实体的属性千差万别。因此，采用传统的关系数据库存储模型不能对各种类型的顾客实体进行统一建模。

本示例程序采用我们提出的可变实体存储模型对顾客实体进行统一建模，使得顾客信息的采集可以应用于不同商家的网上购物系统中，满足商家掌握不同时期的不同顾客信息的要求，并能完成相应的查询功能，做到了顾客信息存储和处理模块的高可重用性和高可伸缩性。

3.2　示例程序的组成及实现

示例程序着重完成顾客信息的采集、存储和查询。为便于讨论，我们仅使用两张表，它们分别是顾客信息表和顾客购物行为表。示例程序在功能上主要包括：修改顾客信息结构、增加顾客信息、顾客信息管理、顾客行为录入和查询。

我们采用 MySQL 数据库＋Tomcat 服务器实现示例程序的三层体系结构，采用 JAXP(Java API for XML Processing)[5] 来解析和处理 XML，采用 Saxon 7.8[6] 来解析和处理 XQuery。

(下转第 75 页)

利用前面所建立的利益函数,计算把分片分布到每个子簇的利益。由于簇 g_1 中包含的站点子簇有: V_1, V_2, V_3, V_4和 V_5, 簇 g_2中包含的站点子簇有: V_6, V_7 和 V_8, 则根据表 1, 我们可得到簇 g_1 和 g_2 的查询代价和更新代价见表 2。要计算簇 g_1的利益,则 $B_{kg1}=\delta(g_0)((q(f_k,g_1)+u(f_k,g_1))-u'(f_k))-u'(f_k,g_0)$, 由于 g_0 已经是顶层, 则 g_0对外更新代价为 0, 即 $u'(f_k,g_0)=0$, 又因为在这一层只有簇 g_1和 g_2, 则可根据表 2 所列数据, 在这一层的更新代价为 20+80, 即 $u'(f_k)=20+80$, 所以 $B_{kg1}=23.2*((180+20)-(80+20))-0=2320$; 同理计算可得簇 g_2的利益 B_{kg2}, 其计算结果可见表 2。

表 1

V	Qry	Upd	Total
V_1	40	10	50
V_2	35	0	35
V_3	25	0	25
V_4	40	10	50
V_5	40	0	40
V_6	25	0	25
V_7	20	40	60
V_8	15	40	55

表 2

G	Qry	Upd	Total	B_{kg}
g_1	180	20	200	2320
g_2	60	80	140	928

根据表 2 所列数据可知, B_{kg1}和 B_{kg2}均大于 0, 即簇 g_1 和 g_2 都有非负的利益, 因此, 分片 f_k 需分布到簇 g_1 和 g_2。在簇 g_1 中, 需要再次决定是将分片分布在簇 g_3还是簇 g_4, 同理可求得簇 g_3的利益 $B_{kg3}=-1533$, 簇 g_4的利益 $B_{kg4}=-1635$, 由于 B_{kg3}和 B_{kg4} 均小于 0, 所以可决定将分片 f_k 分布到最大的利益簇 g_3, 簇 g_4 不再考虑。在簇 g_3 中, 由于簇 g_3 中包含的站点子簇有 V_3, V_4和 V_5, 同样我们可计算把分片分布到每个站点子簇的利益。通过计算可得 $B_{kv3}=-1860$, $B_{kv4}=-1810$, $B_{kv5}=-1830$。由于三个站点均有负的利益函数, 因此可将分片 f_k 分布到最大的利益站点 V_4。在簇 g_2 中再继续往下分, 由于簇 g_2 中包含的站点子簇有 V_6, V_7和 V_8, 同样我们可计算把分片分布到每个站点子簇的利益。通过计算可得 V_6, V_7 和 V_8 均有负的利益函数, 且 V_7 为其中最大的利益站点, 将分片 f_k 分布到最大的利益站点 V_7。到此, 分片 f_k 分布结束, 即分片 f_k 只分布在站点 V_4和 V_7 上。

5　结论

本文在分布式数据库中基于簇的技术基础上提出了分层网络数据分布优化策略, 并给出了优化算法及可行性分析。通过簇的技术, 可以把一个各站点之间通讯代价不同的复杂分层网络映射为一个递归的简单网络, 从而使得求一个复杂分层网络的数据优化分布问题转变为求一个简单网络的数据优化分布问题。特别是对于一般的网络, 给出了把它完全映射到一个分层网络上的方法, 使一般网络的数据分布优化问题也可以利用分层网络的数据分布优化方法来处理, 使得分层网络数据分布优化方法更具实际意义。此外, 这种方法也很容易推广到多权分层网络的数据分布优化问题中。

参考文献:

[1] Huang Y-F, Chen J-H. Fragment Allocation in Distributed Database Design[J]. Journal of Information Science and Engineering, 2001, 17(3): 491-506.

[2] Semeczko G. Using A Double Weighted Clustering Technique for Fragment Allocation in Wide Area Networks[R]. Queensland: Queensland University of Technology, 2000.

[3] Dreyer DR, Overton M L. Two Heuristics for the Steiner Tree Problem[R]. New York: Computer Science Dept, 1996.

[4] 邵佩英. 分布式数据库系统及其应用[M]. 北京:科学出版社, 2000.

(上接第 71 页)

在具体实现时,对于数据库层,作者创建了一个名为 demo 的数据库, 并在其中创建了两张表: 顾客信息表(customer) 和顾客行为表(customer-behavior), 其中顾客信息表的表结构按照可变实体存储模型进行设计。对于应用逻辑层, 我们遵照可变实体处理框架, 将其中的基础功能代码用 JavaBean 来封装, 这些 JavaBean 包括: 处理数据库连接、数据库查询和数据库操作的 Database.java, 验证 XML 文档结构合法性的 SchemaValidatorErrorHandler.java, 解析处理 XML 文档的 HandleXML.java, 以及解析处理 XQuery 的 HandleXQuery.java; 而对于表示层, 为了展示本文所论述的可变实体存储解决方案, 我们编写了以下的 HTML 页面和 JSP 文档: 首页(index.htm)、修改顾客信息结构(updateCustomerStructure.jsp)、增加顾客信息(addCustomer.jsp)、顾客信息管理(updateCustomerInfo.jsp)、顾客行为录入(inputCustomerBehavior.htm) 和查询(query.htm)。

4　总结与展望

本文针对电子商务对数据存储和处理的要求, 提出了一个存储处理可变实体的解决方案, 并从可变实体存储模型和可变实体处理框架两方面介绍了方案的具体内容。该方案通过运用 XML 技术对关系数据库模型进行扩展, 实现了信息的弹性存储, 统一了数据存储处理的接口, 使系统在数据存储方面具有了高可伸缩性和高可重用性。从前面所述的示例程序中可以证明方案是切实可行的。

对于本方案的性能, 与传统关系数据库模型相比, 只有在根据实体属性查询实体数据时, 由于先要从关系数据库中提取数据构造实体集 XML 文档, 再进行查询操作, 因而操作的效率会降低; 而在其他情况下, 由于可以根据实体的 ID 直接使用 SQL 语句进行存取, 两种操作的效率是相当的。

作为本方案的扩展, 可以把实体的外码集也用 XML 来描述, 这样所有的实体都可以用相同的结构来描述, 即(实体 ID, 外码集, 属性集)。此时, 不仅实体的相互关系可以做到弹性变化, 而且也可以统一对实体数据的操作。

参考文献:

[1] 王京. 随需应变的电子商务——记 IBM 的 IT 革命[J]. 商场现代化, 2003, (3).

[2] W3C. XML 1.0 (Third Edition)[S/OL]. http://www.w3.org/TR/2003/PER-xml-20031030, 2004.

[3] W3C. XML Schema (Part 0、Part 1、Part 2)[S/OL]. http://www.w3.org/TR/2001/, 2001.

[4] W3C. XQuery 1.0: An XML Query Language[S/OL]. http://www.w3.org/TR/2003/, 2003.

[5] Harold ER. Processing XML with Java[M]. Addison-Wesley, 2003.

[6] Kay MH. Saxon — The XSLT and XQuery Processor[Z/OL]. http://saxon.sourceforge.net/, 2003.