

树形结构数据在数字矿山中的存储管理与应用

张维国,孙效玉,周 冲,董 波

(东北大学,辽宁 沈阳 110819)

摘 要:在矿山信息化建设过程中需要建立高效便捷的数据管理和更新体系,其中露天矿组织管理机构是一种典型的树形图结构。针对树形结构数据在关系型数据库中的优化存储管理和交互操作较难实现的问题,文中设计了简化的单一关系数据库表结构,采用数据库高性能嵌套查询的公用表表达式语句,提出了递归数据绑定的树形结构数据管理模式,实现了树形结构数据的高效管理,并结合 C#开发语言的树形控件及矿山组织管理机构的实际情况,在数字矿山建设的工作任务管理系统中得到了实际应用。

关键词:数字矿山;关系数据库;树形结构;嵌套查询;公用表表达式;树形控件

中图分类号:TP39

文献标识码:A

文章编号:1673-629X(2015)03-0150-04

doi:10.3969/j.issn.1673-629X.2015.03.034

Storage Management and Application of Tree Structure Data in Digital Mine

ZHANG Wei-guo, SUN Xiao-yu, ZHOU Chong, DONG Bo

(Northeastern University, Shenyang 110819, China)

Abstract: In the process of mine informatization construction, it is necessary to establish efficient and convenient system of data management and updating, among which the open-pit mine organization management structure is a typical tree graph structure. In order to realize more difficult problems in relational database of optimized storage management and interactive operation of tree structure data, design a simplified single relational database table structure and adopt the common table expression statements of the high performance nested query, put forward a data management mode of tree structure with recursive data binding, finally realize the efficient management of tree structure data, which is verified by the application of mining task management system combined with Treeview control of C# and the practical situation of mining organization and management agencies in digital mine.

Key words: digital mine; relational database; tree structure; nested query; common table expression; Treeview control

0 引言

在数字矿山建设过程中,经常遇到组织管理机构、工作任务类型、设备故障分类等呈现树形结构数据的数据管理问题。树形结构是数据理论中用来描述复杂的层次关系时使用的一个重要概念^[1]。树形结构管理具有条理清晰、结构简单、操作方便等特点^[2]。

在软件开发过程中,树形结构数据在关系数据库存储的数据结构和管理方式通常有两种途径:一是通过编写应用程序代码,逐条递归条件查询单一的数据表,并按照数据内容重新形成树形结构视图;二是创建多个关系数据表,进行多表联合查询得到树形结构数

据表。具体途径可以通过树^[3-5]、半结构化数据^[6]、关系表^[7]和存储过程^[8]、XML 结构^[2]、自定义数据结构^[8]等多种方式实现存储。但是以上各种数据结构存在复杂、不宜维护、编程实现较困难和通用性差的缺点。此外,VB、VC 等高级语言均提供了功能强大的树形视图控件,可以用来描述复杂的层次关系^[9-12],在具体编程实现上也不尽相同。当前流行且较通用的 C# 语言提供了强大的树形控件(Treeview),但是存在无法对树形节点定位、数据动态绑定困难的问题。

文中提出了简化的单一关系数据库表结构、高效的嵌套子查询、递归数据绑定的树形结构数据管理模式。

收稿日期:2014-04-02

修回日期:2014-07-06

网络出版时间:2015-01-20

基金项目:科技部“十二五”科技支撑计划项目(2012BAJ17B01)

作者简介:张维国(1981-),男,博士研究生,研究方向为数字矿山工程、矿山管理信息系统建设和数据挖掘等;导师:吴立新,博士生导师,教育部长江学者特聘教授,研究方向为遥感-岩石力学、三维地学建模、矿山开采沉陷与控制、GIS 理论与算法、环境与灾害遥感、数字矿山、数字城市、矿区可持续发展等。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20150120.2159.012.html>

式。在实际工程应用中,结合矿山工作任务管理系统开发,实现了无限分层树形结构数据简便高效的管理。

1 数据库设计与嵌套查询方法

1.1 数据表结构设计及内容

树形结构图的节点数据可以看作是当前节点与上级节点具有直接的关联关系,其子节点的多少和后续关系与该节点的数据层次无关。因此树形结构数据表可以简化设计为:标识数据表中唯一性的主键、当前节点的父节点标识、节点名称和节点其他关联属性等构成的单一关系数据表。数据表结构设计如表 1 所示。

表 1 数据库表结构设计

列名	含义	数据类型	空/非空	说明
ID	主键标识	int	非空	自增
Parent	父节点标识	int	空	来自 ID
Name	节点名称	nvarchar(50)	空	节点名
Type	节点类型	nvarchar(10)	空	关联图片与操作
FKID	外键标识	nvarchar(50)	空	关联其他表内容

1.2 递归查询的公用表表达式

查询关系数据库中上述树形结构的数据,需要从某一节点查询该节点之后的所有子节点,遍历整个树形结构。借助数据库的结构化查询语言 (Structured Query Language, SQL),可以解决应用程序复杂性、提高执行效率等问题。在 SQL Server 的早期版本中,递归查询通常需要使用临时表、游标和逻辑来控制递归步骤流。比如,SQL 语句实现遍历整个树形结构,可以用 IN 操作符多次调用查询语句实现嵌套查询,但嵌套层次过多会使得 SQL 语句非常难以阅读和维护;此外,虽然可以采用表变量的方式解决,但是表变量会使嵌套查询语句更复杂,表变量使用的临时表也会增加额外的 I/O 开销带来性能的损失。

为了解决以上问题,可以通过 SQL Server 2005 及更高版本的公用表表达式 (Common Table Expression, CTE) 来解决分层嵌套查询的问题。CTE 是在单个 SELECT、INSERT、UPDATE、DELETE 或 CREATE VIEW 语句的执行范围内定义的临时结果集^[13]。CTE 与派生表类似,具体表现在不存储为对象,并且只在查询期间有效。与派生表的不同之处在于,CTE 可自引用,还可在同一查询中引用多次^[14]。CTE 通常在创建递归查询或者替换视图时使用。CTE 不必将定义存储在元数据中,它能够启用按从标量嵌套 SELECT 语句派生的列进行分组,或者按不确定性函数或有外部访问的函数进行分组,同时,在同一语句可以多次引用生成的表。

CTE 的结构由 CTE 的表达式名称、可选列列表和定义 CTE 的查询组成。定义 CTE 后,可以如表或者视

图一样对其进行引用,作为定义 SELECT 语句的一部分。

CTE 的基本语法结构如下:

```
WITH expression_name [ ( column_name[ ,... n] ) ]
AS
( CTE_query_definition )
```

CTE 语法结构只有在查询定义中,只有对所有结果列都提供了不同的名称时,列名称列表才是可选的。

CTE 的外部查询语句的语句如下:

```
SELECT <column_list>
FROM expression_name;
```

使用 CTE 的优点之一是可以获得高可读性和低维护量的复杂查询。查询可以分为单独块、简单块、逻辑生成块。这些简单块可用于生成更复杂的临时 CTE,直到生成最终结果集。CTE 的另外一个优点是能够引用其自身,从而创建递归 CTE。当某个查询引用递归 CTE 时,它即被称为递归查询,递归查询通常用于返回分层数据。递归 CTE 是一个重复执行初始 CTE,以返回数据子集直到获取完整结果集的公用表表达式。

递归 CTE 与其他编程语言的递归方法一样,但在数据库数据查询过程中得到性能的优化。CTE 递归查询的执行过程如下^[15]。

Step1: 定义定位点成员。

递归 CTE 至少需要包含两个查询,首先需要定义一组查询得到 CTE 结构的基准结果集,该查询语句是由一个或者多个 UNION ALL、UNION、EXCEPT 或 INTERSECT 运算符连接的查询,若查询未引用 CTE 自身,则该查询定义为定位点成员。定位点成员是一个返回有效表的查询,是递归的基础。所有的定位点成员需要放在 CTE 递归查询成员之前,最后一个定位点成员必须使用 UNION ALL 运算符连接其后的第一个递归成员。

Step2: 递归调用 CTE 自身。

递归调用包括一个由引用 CTE 本身,并由 UNION ALL 运算符连接查询定义的递归成员,即最后一个 UNION ALL 运算符连接的查询。这种引用不同于外部查询中对 CTE 名称的引用,它是触发递归的关键元素,在 CTE 结果表产生前调用。递归调用第一次由定位成员返回初始结果集,之后由递归成员和 CTE 自身引用调用后续查询结果集。

Step3: 终止条件检查。

终止条件检查是隐式的,只有上一个调用未返回行时,递归才会停止。当数据中存在循环或者代码本身存在逻辑死循环时,可使用 MAXRECURSION 操作符限制递归成员的调用次数,该操作符紧接在 CTE 外

部查询后,指定语法为:“外部查询 OPTION (MAXRECURSION n) ”。

1.3 树形结构数据管理

按照 CTE 的结构短语定义和执行过程,树形结构数据的递归查询实现短语如下:

```
WITH cteStructure AS
(
    SELECT[ ID ], [ Parent ], [ Name ]
    FROM[ Treeview ] WHERE [ ID ] = 1 / * 输入节点主键序号
    */
    UNION ALL
    SELECT
    u. [ ID ], u. [ Parent ], u. [ Name ]
    FROM[ Treeview ] u
    INNER JOINcteStructure
    ON u. [ Parent ] = cteStructure. [ ID ]
)
```

以上 CTE 在执行时先得到定位点成员,即由根节点的主键查询得到的结果集,之后循环引用 CTE 名称查询下一级结果集,直至递归成员返回的结果集为空时,递归结束。

数据库的增加和更新直接用 INSERT、UPDATE 更改表内容即可,而对于树形结构数据的删除,依然采用 CTE 递归查询将要删除节点及下层关系内容节点的主键序号,在定义 CTE 结构短语后,直接调用删除语句,其具体删除语句如下:

```
DELETE [ StructureTreeview ]
WHERE [ StructureTreeview ]. [ ID ]
IN ( SELECT [ StructureID ] FROM cteStructure )
```

2 数据绑定及树形控件处理

2.1 动态数据绑定

ADO.NET 可以直接调用 CTE 递归短语查询数据表内容,也可以将 CTE 实现语句封装成存储过程。查询返回的结果表作为 Treeview 控件数据绑定的数据源,按照数据结构关系逐条在 Treeview 控件添加树形节点。

调用 CTE 的 C#语句如下:

```
DataGridView dataGridView = new DataGridView( dataTable );
dataGridView. RowFilter = " StructureParent = 1 ";
TreeNodeDataBind( dataGridView, this. treeViewStructure. Nodes.
Add( “ 矿山 ” ) );
```

其中,dataTable 是通过 CTE 查询到的树形结构数据的表结构内容,实现代码中 TreeNodeDataBind() 函数是 Treeview 控件数据绑定函数,数据绑定函数实现如下:

```
public void TreeNodeDataBind( DataGridView dataGridView, TreeNode
parentNode )
```

```
{
    TreeNode treeNode = new TreeNode( );
    foreach( DataGridView dataGridView in dataGridView )
    {
        //添加并设置新节点属性
        treeNode = parentNode. Nodes. Add( dataGridView [ " Name " ].
        ToString( ) );
        treeNode. Tag = dataGridView [ " ID " ]. ToString( ) + ", " + da-
        taGridView [ " Type " ]. ToString( );
        //查询视图中当前节点下的子节点
        dataGridView. RowFilter = " Parent = " + dataGridView [ " ID " ]. ToS-
        tring( );
        TreeNodeDataBind( dataGridView, treeNode );//递归调用添加节-
        点
    }
}
```

2.2 树节点编辑操作

树形控件的编辑操作分为节点移动和复制、节点名称修改、节点添加和删除等内容,这些操作有一个共同点是要对鼠标点击的节点进行定位,属于树形控件的复杂操作^[16]。但 Treeview 控件在鼠标点击事件中无法由事件属性值直接返回 Node 节点的对象,经常会出现添加和复制的节点置于根节点、错误删除节点等问题。解决这些问题可以由 Treeview 控件点击事件 TreeNodeMouseClickEventArgs 对象中的坐标值查询鼠标所在节点的位置,即由 Treeview 控件的 GetNodeAt() 方法得到鼠标点击的节点对象,代码如下:

```
TreeNode currentNode = this. treeViewStructure. GetNodeAt
( new Point( e. X, e. Y ) );
```

得到 Treeview 控件鼠标点击的节点后,需要设置 Treeview 控件的已选择节点属性:

```
this. treeViewStructure. SelectedNode = currentNode;
```

如果使用右键弹出 Treeview 控件的内容菜单操作相应的节点,需要在菜单操作事件中重新得到已选择的节点对象:

```
TreeNode treeNode = this. treeViewStructure. SelectedNode;
```

数据绑定时,在 Treeview 控件节点的 Tag 属性中加入了节点类型内容,节点类型可作为 Treeview 控件弹出不同内容菜单的判断依据。

2.3 数据更新实现

添加数据:应用程序代码首先判断执行方法是插入同一级节点还是子节点,然后通过 Treeview 节点 Tag 属性中的主键 ID,插入数据库新的节点内容,新节点 Parent 内容为鼠标点击节点的主键 ID。

修改节点:由鼠标点击事件触发的 Node 对象得到节点主键 ID,通过 ID 直接更新关系数据库中对应节点的内容。

删除节点:判断 Treeview 控件鼠标点击节点下是

否有子节点,如果存在子节点,系统提示用户是否删除。删除节点操作可调用 CTE 递归查询短语,得到删除节点下所有的子节点的结果集,并执行 DELETE 语句,实现子节点的快速删除。

3 树形结构数据应用实例

矿山组织管理机构具有金字塔形^[17],即树形结构分层的模式,按照职责和管理内容划分职能部门和生产管理科室。在实际应用中,可能出现同一个人兼多个职责的情况,因此,无法通过树形节点的名称定位部门或者人员的层次关系。此外,组织管理机构随着矿山生产任务的变化和人员升迁的变动,需要经常更新树形结构图。组织管理机构树形图必须按照实际管理结构由树形控件展示具体的结构层次、直观快速定位生产部门或者人员的节点位置,以便第一时间指定待执行的任务和工作流。另外,为了适应树形组织管理机构的变化,采用动态数据绑定技术实现数据库树形结构数据的动态查询、更新、删除以及复制、移动等功能。

以某一典型矿山的实际组织管理机构为例,建立树形结构数据的数据库存储结构和内容,由 .NET 平台的 ADO.NET 技术与树形结构数据所在的数据源进行连接,通过 C#语言的 Treeview 控件实现数据动态绑定和管理操作。基于上述树形结构数据管理模式开发的矿山工作任务管理系统的应用开发证实,以上编程实现模式能够快速定位部门或人员的节点位置,制定工作任务内容或查询任务执行的详细信息。

4 结束语

树形结构数据所反映的分层关系,在数字矿山建设中具有广泛的应用。文中结合矿山组织管理机构树形结构图的工程实例,设计了简单清晰的数据库表结构,采用 CTE 公用表表达式实现高效递归无限分层查询。

(1)解决了树形结构数据在关系型数据库中存储复杂和通用性差的问题;

(2)解决了 C#语言中树形控件的数据动态绑定以及数据交互操作中节点定位困难的问题;

(3)通过关系型数据库中树形结构数据子节点与父节点主键的关联设计,解决了树形控件中多个同名节点数据绑定和定位的问题;

(4)提出了树形结构数据管理过程中的通用编程实现模式,在数字矿山建设和类似工程项目编程中可以得到推广应用。

实际应用表明,采用以上途径能够很好地解决诸如组织管理机构相似树形图的优化存储和高效管理的问题。

参考文献:

- [1] 朱月香. TreeView 控件的应用[J]. 兵工自动化,2003,22(6):54-55.
- [2] Cebeci Z, Erdogan Y. Tree view editing learning object meta-data[J]. Interdisciplinary Journal of E-learning and Learning Objects,2005,1(1):99-108.
- [3] 特日根,李巍,李雄飞. 动态有序树存储模型与实现方法[J]. 计算机研究与发展,2013,50(5):969-985.
- [4] 陆霞. 基于二叉键树的多模式匹配算法的研究[J]. 电脑知识与技术,2010(15):4302-4304.
- [5] 孙英晖,田少鹏. 基于多叉树结构的号码存储方法[J]. 指挥信息系统与技术,2011,2(1):66-69.
- [6] 张雨佳,苏中滨,吴华瑞,等. 半结构化数据的动态树存储模型研究[J]. 计算机应用与软件,2011,28(5):86-90.
- [7] 洪伟,吴云,周国祥. 基于树形结构的汽车试验集成系统数据库设计[J]. 计算机技术与发展,2008,18(2):177-179.
- [8] 浮光宾,王葵如,张明伦. ASP.NET 中 TreeView 控件的动态绑定及定位展开[J]. 计算机系统应用,2008(6):112-115.
- [9] 储岳中. 基于递归算法和树形控件的动态树形图的实现[J]. 计算机技术与发展,2007,17(6):87-89.
- [10] 李俊锋,方明. 基于编码的 TreeView 控件节点生成算法[J]. 电脑知识与技术,2009,5(4):847-848.
- [11] 葛斌. VC++6.0 自动创建树形结构[J]. 电脑编程技巧与维护,2008(10):17-18.
- [12] 畅育超. TreeView 控件基础与综合应用[J]. 电脑编程技巧与维护,2013(11):48-52.
- [13] Burzanska M, Stencel K, Wisniewski P. Pushing predicates into recursive SQL common table expressions[C]//Proc of advances in databases and information systems. Berlin:Springer,2009:194-205.
- [14] Przymus P, Boniewicz A, Burzańska M, et al. Recursive query facilities in relational databases: a survey[C]//Proc of database theory and application, bio-science and bio-technology. Berlin:Springer,2010:89-99.
- [15] Ben-Gan I, Kollar L, Sarka D, et al. Inside Microsoft® SQL Server® 2008: T-SQL Querying[M]. [s.l.]: O'Reilly Media, Inc,2009:327-330.
- [16] 于晓静,管建和. TreeView 控件复杂操作的编程技巧[J]. 电脑编程技巧与维护,2007(4):25-28.
- [17] 吴立新,殷作如,邓智毅,等. 论 21 世纪的矿山—数字矿山[J]. 煤炭学报,2000,25(4):337-342.

树形结构数据在数字矿山中的存储管理与应用

作者：[张维国](#)，[孙效玉](#)，[周冲](#)，[董波](#)，[ZHANG Wei-guo](#)，[SUN Xiao-yu](#)，[ZHOU Chong](#)，[DONG Bo](#)
作者单位：[东北大学, 辽宁 沈阳, 110819](#)
刊名：[计算机技术与发展](#)
英文刊名：[Computer Technology and Development](#)
年，卷(期)：2015(3)

引用本文格式：[张维国](#). [孙效玉](#). [周冲](#). [董波](#). [ZHANG Wei-guo](#). [SUN Xiao-yu](#). [ZHOU Chong](#). [DONG Bo](#) [树形结构数据在数字矿山中的存储管理与应用](#) [期刊论文] - [计算机技术与发展](#) 2015(3)