软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(9):2290−2302 [doi: 10.13328/j.cnki.jos.004946]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

偏序时态 XML 索引 TempPartialIndex*

汤 娜, 叶小平, 汤 庸, 彭 鹏, 杜梦圆

(华南师范大学 计算机学院,广东 广州 510631)

通讯作者: 汤庸, E-mail: sinceretn@qq.com

摘 要: 时态数据管理是常规数据管理的深化和扩展,具有理论研究的意义与实践应用的价值.时态数据索引是时态数据管理的重要技术支撑,是其中的一个研究热点.首先,提出了一种时态数据结构,通过数据节点间的偏序关系,可将常规的二维时间区间的处理转化为基于偏序的时态等价类上的一维的处理,该数据结构可以快速有效地处理时态操作;其次,在该新型时态数据结构基础上研究了时态 XML 索引 TempPartialIndex,其基本特征是将时态数据结构整合到非时态的 XML 索引中,即,将其整合到语义层之中,通过时态过滤和语义过滤掉大量节点之后,再进行结构连接;另外,着重讨论了基于 TempPartialIndex“一次一集合”及其时态变量查询和增量式的动态更新机制.同时,仿真结果表明:TempPartialIndex 能够有效地支持时态 XML 的各类查询及更新操作,技术上具有可行性和有效性.

关键词: 时态偏序关系;时态数据结构;时态 XML 索引;时态变量查询;增量式动态更新

中图法分类号: TP311

中文引用格式: 汤娜,叶小平,汤庸,彭鹏,杜梦圆.偏序时态 XML 索引 TempPartialIndex.软件学报,2016,27(9):2290−2302. http://www.jos.org.cn/1000-9825/4946.htm

英文引用格式: Tang N, Ye XP, Tang Y, Peng P, Du MY. Temporal XML index based on temporal partial-order relationship. Ruan Jian Xue Bao/Journal of Software, 2016,27(9):2290−2302 (in Chinese). http://www.jos.org.cn/1000-9825/4946.htm

Temporal XML Index Based on Temporal Partial-Order Relationship

TANG Na, YE Xiao-Ping, TANG Yong, PENG Peng, DU Meng-Yuan

(School of Computer Science, South China Normal University, Guangzhou 510631, China)

Abstract: Temporal data management is an extended field of data management, and the research on this field has theoretical significance and practical application value. The index of temporal data is crucial to temporal data management and thus is one of the hot research topics. First, based on the partial-order mathematical relationship, this paper proposes a data structure which can convert a temporal query processing of two-dimension into one-dimension. This structure can process all the temporal relationship operations efficiently. Secondly, this paper presents an indexing scheme called TempPartialIndex which combines temporal partial-order data structure into non-temporal XML index. In TempPartialIndex schema, the processing of query begins with mapping and filtering with the semantic and temporal constraints so that a large number of nodes will be filtered out and only a few nodes will be left. Then a structural join algorithm is executed on the left nodes. Furthermore, the paper discusses the query based on set-at-a-time, the temporal variable query and modification. Simulations show that TempPartialIndex can process the temporal XML query and update efficiently.

Key words: temporal partial-order relationship; temporal data structure; temporal XML index; temporal variable query; incremental dynamic modification

* 基金项目: 国家高技术研究发展计划(863)(2013AA01A212); 国家自然科学基金(60970044, 61272067); 广东省自然科学基金(S2011010003409); 广东省自然科学基金团队研究项目(S2012030006242)

Foundation item: National High-tech R&D Program of China (863) (2013AA01A212); National Natural Science Foundation of China (60970044, 61272067); Natural Science Foundation of Guangdong Province of China (S2011010003409); Natural Group Research Projects Foundation of Guangdong Province of China (S2012030006242)

收稿时间: 2015-06-08; 修改时间: 2015-09-19; 采用时间: 2015-11-10

XML 作为一种广泛应用的自定义性的元语言,已经成为 Web 信息交互和集成的标准.时间是客观事物的基本属性,XML 数据随时间变化是 XML 数据管理的一种常态.时态 XML 数据管理可分为两种情形.

(1) 以文档为中心,着重考虑 XML 文档的变化过程.

由于 XML 文档结构框架和数据形式本身会随着时间变化,其变迁历史需要被记录并提供相应的版本查询,这就是 XML 配置的版本管理,其对时间的考虑定位在文档自身变化层面,不涉及文档中数据内容的时间变化.这类文档的结构性不强.这类文档的时态查询主要是查询哪些文档在给定时间区间包含了某些特定关键词,查询的信息中不包括结构信息.文献[1,2]就是针对这类时态 XML 文档所做的时态 XML 倒排索引.

(2) 以数据为中心,着重考虑文档中数据内容随时间变化的情形.

所有的对象在现实世界中都有生存期和有效时间,为了记录对象的有效性和对数据历史信息回溯检索,对象时态信息的描述不可或缺,此时需要考虑数据本身的语义和数据之间的结构信息[3].通过人们的研究工作,现在 XML 可以很好地支持 temporally grouped[4]的符合时态数据特性的数据模型;同时,XML 的查询语言 XQuery 具有图灵完备和自身可扩展(natively extensible)的特性[5,6],可以自定义很多复杂的时态查询,所以, XML 成为支持时态数据的表达、查询的平台,例如基于 XML 的电子病历.本文主要讨论这类基于数据的时态 XML 查询优化.

基于数据的时态 XML 查询通常包括 4 种约束信息:语义、结构、值和时态约束条件.快速地查询一个半结构化的时态 XML 文档,是一个极具挑战和复杂的工作.研究者往往通过将文档映射到关系数据库或面向对象的关系数据库中[7]、对时间区间划分(slicing)[8]、提出时态 XML 索引结构等手段来提高查询的效率.在研究过程中学者发现,采用一个非时态的 XML 索引来计算某个给定时间范围内的合法路径是相当耗费时间[9].例如, Amagasa 等人[10]提出了一种用有效路径建立索引的方法,并采用时态关系数据来存放有效路径.但由于索引完全没有反映时态信息,每次需要从头到尾地扫描时态关系数据表以找到符合时态约束的有效路径,因而索引效率不高.为获得良好的查询性能,将时态信息整合到非时态的 XML 索引中是时态 XML 查询优化的基本要素之一.时态信息整合的工作主要有两种基本类型:

类型(1)将时态信息整合到索引当中,如:文献[11]在 FIX 索引的基础上,将节点时态信息作为特征向量区间$[\lambda_{\min},\lambda_{\max}]$的补充信息存储在 B 树索引中,查询处理首先根据特征向量值进行过滤筛选,符合条件的片段再进行时态筛选.类似的,文献[12]则将时态信息单纯地整合到 UB-tree 中.这类工作的特点是将时态信息整合到 XML 索引中,但是并没有针对时态信息结构和时态运算的特点来建立相应的时态数据结构,而且时态信息只是作为 XML 的一个属性来进行处理;

类型(2)根据时态数据的特点创建了适合时态关系运算的数据结构,通过设计时态数据结构提高了时态运算的效率,然后,将时态数据结构整合到 XML 索引之中[9,13−15].由于非时态的 XML 的查询处理中,结构约束的处理难度最高,而且语义约束和结构约束是必然给出的约束条件,值约束是可选的,所以非时态的 XML 索引的查询处理会优先处理结构和语义约束,而值约束的处理是最后进行的.类型(2)的工作将时态约束的匹配和过滤作为一种特殊的值约束进行处理,往往在处理的最后一步完成.查询结果是满足时态约束的节点集和与满足所有其他约束的节点集的交集.这类工作虽然创建了时态数据结构来优化和提高时态运算的效率,但是并没有针对如何将时态数据结构整合到 XML 索引和查询处理中进行优化,只是单纯地将时态约束的匹配和过滤作为 XML 文档的值约束来处理.

事实上,由于时态 XML 数据中每个节点都具有时间标签,数据的时态过滤在整个数据过滤过程中占有比较大的比重,先行进行时态过滤可以极大地减少中间过程的工作量.为了进一步提高时态 XML 索引的查询效率,本文首先提出一种基于时态偏序的时态数据结构,该结构的基本特征是将在二维时间区间上的时态关系操作转换为基于偏序的时态等价类中的一维操作,提高了时态关系操作的效率;然后,将时态数据结构整合到 XML 索引结构的语义层之中,不再只是把时态约束的匹配和过滤作为 XML 文档的值属性来处理.通过借鉴关系代数中“先简(选择和投影)后繁(连接)”的优化思想,改进和优化了时态数据结构与非时态 XML 索引整合和查询处理的方式,即:将所有时态数据结构整合到 XML 的语义信息中,将相同语义标签的数据划分成若干个基于偏序的

时态等价类,先通过进行语义和时态处理来过滤掉大量节点,再判断剩余的节点是否满足结构约束,因此节约了查询处理的时间开销.当查询信息不存在时态约束时,其性能等价于一个非时态的 XML 索引.

当表达对象在现实世界的有效时间时,往往都没有明确的截止时间,因此在时态数据库中引入了 Now 变量. Now 的语义和查询处理都是时态数据库相关研究中的一个难点[16],文献[17]讨论了 Now 的基本语义以及包含 Now 的时态操作和代数系统.但现有的时态 XML 索引没有对 Now 变量做特殊处理来提高基于 Now 的时态操作的效率.本文通过扩展基于时态等价关系的偏序结构,使其支持 Now 变量的处理,并能快速地处理时态变量的快速查询和更新.

本文提出了一种新颖的基于时态偏序关系的时态数据结构,该结构可以将二维的时间区间上的时态关系操作转化为基于偏序的时态等价类中的一维时间点的处理,支持快速地处理各类时态关系操作;研究基于上述时态数据结构的时态 XML 索引 TempPartialIndex,该索引通过优化配置各类约束条件而实现相应的时态 XML 查询优化;讨论了基于索引的一次一集合的常规时态 XML 数据查询和基于 Now 的时态变量查询,同时研究了索引的增量式动态更新机制.

本文第 1 节讨论基于偏序的时态数据结构和建立于其上的时态索引 TempPartialIndex.第 2 节讨论基于 TempPartialIndex 的常规时态查询和动态更新算法.第 3 节研究基于时态变量 Now 的查询机制.第 4 节是相应的实验仿真.

1 TempPartialIndex 索引模型

1.1 时态数据结构

时态数据节点的有效时间用 *VT* 来表示,*VT*=[*VTs*,*VTe*],*VTs* 和 *VTe* 分别表示 *VT* 的始点和终点(*VTs*<=*VTe*). *Vstart*(*v*)和 *Vend*(*v*)分别表示节点 *v* 的 *VTs* 和 *VTe*.如果一个节点当前有效,但没有确定的有效截止时间,则 *Vend*(*v*)=*Now*,如果一个节点有确定的有效截止时间,则 *Vend*(*v*)<>*Now*,即,*Vend*(*v*)≠*Now*.例如:张三是我校的员工,其在校工作的有效时间从 2010 年开始,目前仍在我校工作,则 *Vstart*(*v*)=2010,*Vend*(*v*)=*Now*.如果张三 2014 年离开了我校,则 *Vend*(*v*)=2014,即,其 *Vend*(*v*)值不等于 *Now*.我们将在第 3 节更为系统地讨论 *Now* 变量的处理方法.*Vspan*(*v*)表示节点 *v* 的有效时间跨度,如果 *Vend*(*v*)≠*Now*,则 *Vspan*(*v*)=*Vend*(*v*)−*Vstart*(*v*).

定义 1(**时态等价关系**). 对于时态 XML 文档中的两节点 *u* 和 *v*,若 *Vend*(*u*)以及 *Vend*(*v*)均不等于 *Now* 时,如果 *Vspan*(*u*)=*Vspan*(*v*),则称 *u* 和 *v* 之间存在着时态等价关系,记为(*u*)≡(*v*).

定理 1. 若时态 XML 文档中的两节点 *u* 和 *v* 存在(*u*)≡(*v*),则≡是满足自反性、对称性和传递性的等价关系.

定义 2(**时态偏序关系**). 对于时态 XML 文档中的两节点 *u* 和 *v*,如果 *Vstart*(*u*)<*Vstart*(*v*),则称 *u* 和 *v* 之间存在着时态偏序关系,记为(*u*)«(*v*).

定理 2. 对于时态 XML 文档中的两节点 *u* 和 *v*,如果两个节点之间的关系满足反自反性和传递性,则两个节点之间的关系是一种拟序关系,定义 2 中两个节点之间的关系«是一种拟序关系.

定义 3(**基于时态等价关系的等价类**[***Flag***],**简称时态等价类**). 节点 $v_1,v_2,\ldots,v_n$,如果所有节点 $v_i(1<=i<=n)$的 *Vend*(v_i)<>*Now*,且有 *Vspan*(v_1)=*Vspan*(v_2)=…=*Vspan*(v_n),则节点 $v_1,v_2,\ldots,v_n$ 属于同一个等价类[*Flag*],[*Flag*]记为[*Vspan*(v_i)].

例 1(时态等价类的实例):若 5 个节点 *u* 和 *v* 的 *VT* 分别为{[100,150],[50,70],[50,170],[120,170],[100,120]},则可以划分为 3 个等价类.

[*Flag*]	等价类中包含的节点
[20]	{[50,70],[100,120]}
[50]	{[100,150],[50,170]}
[120]	{[50,170]}

定理 3(**时态等价类的时态属性**). 对于任意两个时态等价类[$Flag_1$]和[$Flag_2$]及任意一个有效时间区间 *VI*,存在如下属性:

(1) 任意的两个节点 u 和 v,u 节点属于时态等价类[$Flag_1$],v 节点属于时态等价类[$Flag_2$],如果 $VT(u)\subseteq VT(v)$,则 $Flag_1<=Flag_2$;

(2) 如果 $Flag_1\neq Vspan(VI)$,则在时态等价类[$Flag_1$]中不存在一个节点 w,其 $Equals(VI,VT(w))$=true;

(3) 如果 $Flag_1>Vspan(VI)$,则在时态等价类[$Flag_1$]中不存在一个节点 w,其时态区间关系 $Contains(VI,VT(w))$=true,$Started\_by(VI,VT(w))$=true,$Finished\_by(VI,VT(w))$=true;

(4) 如果 $Flag_1<Vspan(VI)$,则在时态等价类[$Flag_1$]中不存在一个节点 w,使得时态区间关系 $During(VI,VT(w))$=true,$Starts\ (VI,VT(w))$=true,$Finishes(VI,VT(w))$=true.

关系 *Equals*(),*Contains*(),*Started_by*(),*Finished_by*(),*During*(),*Starts*(),*Finishes*()均为 Allen 的 13 种时态运算关系[18]中的时态关系.

1.2 节点编码及关系判定

在处理时态 XML 时,为了反映节点间关联,通常需要对节点进行编码.本文采用了 Dewey 前缀编码[19].

定义 4(前缀编码). 将一个节点的父亲节点的编码作为该节点编码的前缀,则节点的前缀编码定义如下:

(1) r 是根节点,则 r 的前缀编码 $code(r)$=0;

(2) 设 $v_1,v_2,...,v_n$ 是 r 的子节点,则 v_i 的前缀编码 $code(v_i)=code(r).i(1<=i<=n)$;

(3) 设第 k 层(k>1,根节点所在的层为 1 层)的节点 v 有一条入边,入边的父节点是 u,v 是 u 的第 i 个子节点,则 v 的前缀编码为 $code(v)=code(u).i$,$code(u)$称为 v 的前缀码.

例 2(时态 XML 数据及其前缀编码实例):在图 1 中,根节点 Company 的前缀编码为 $code(r)$=0,根节点没有前缀码;第 2 层中编号为 5(此编号是为了方便说明,没有实际意义)的 Employee 节点的前缀编码为 $code(5)$=0.1,编号为 95 的 Branch 节点前缀编码为 $code(95)$=0.3,Employee 节点和 Branch 节点的前缀码均为 0;第 5 层的编号为 65 的叶节点的前缀编码为 $code(65)$=0.2.2.1.1,其前缀码为 0.2.2.1.在图 1 中,节点的有效时间通过对该节点的所有出边的有效时间做并运算得到.编号为 70 的节点有两个出边,所以该节点的有效时间为[2000,20004]$\subseteq$[2004,*Now*)=[2000,*Now*),即,$Vstart(70)$=2000,$Vend(70)$=*Now*.

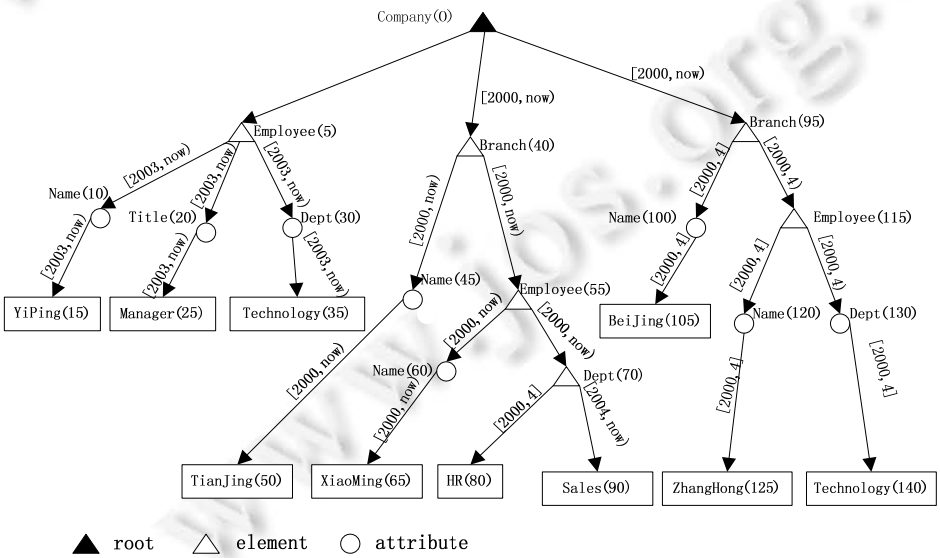

Fig.1 Instance of temporal XML document data model

图 1 时态 XML 文档数据模型实例

定理 4. u 是 v 父节点的充要条件是 v 的前缀码等于 u 的前缀编码;u 是 v 祖先节点的充要条件是 v 的前缀码包含 u 的前缀编码.

前缀编码的一个重要性质是它们的字典有序:以节点 r 为根的子树中的任意一个节点 u,它的前缀编码 $code(u)$大于(小于)它的左(右)兄弟子树中的所有节点的前缀编码.因此,前缀编码不仅能够有效地支持包含关系的计算,而且能够有效地支持文档位置关系的计算.

算法 1. 基于 Hash 表的祖先关系判定.

假设 *ListA*,*ListB* 分别存储了语义标签为 *A* 和 *B* 的节点的前缀编码

功能:查询 ListA 中的前缀编码表示的节点是否在 *ListB* 中存在祖先节点.

输入:*ListA* 和 *ListB*;

输出:节点列表 *ResultList*,*ListB* 中其祖先存在于 *ListA* 中的节点.

Step 1. 将 *ListA* 中的前缀编码进行 hash 映射,每个前缀编码都得到一个 hash 地址;

Step 2. 对 *ListB* 中的每一个前缀编码进行判断:截取出该前缀编码的所有前缀码,判断 Step 1 中的 Hash 地址中是否存在这些前缀码的 hash 地址,若存在,根据定理 4,将此前缀码加入到 *ResultList* 中.

经过节点过滤后,假设 *ListB*(子孙层)有 Q 个节点,*ListA*(祖先层)有 W 个节点,如果不采用一定的策略,性能是映射中的一个大问题,则祖先/子孙匹配效率会非常低,甚至低至 $O(Q\times W)$;如果祖先层没有什么重复的节点,效率会低达 $O(Q^2)$.而本文采用前缀编码结合 hash 映射的方法,取代了对“值”的缓慢搜索.通过时间复杂度为 $O(1)$就可以将某个查询值对应的节点对象找出.由于子孙层有 Q 个节点,通过子孙节点的前缀编码得到祖先节点的前缀编码,再通过 hash 映射判断祖先层中是否存在着此前缀编码对应的节点,所以算法的效率为 $O(Qh)$,其中,h 为树的高度.

1.3 时态XML索引TempPartialIndex模型

在每个时态等价类[*Flag*]中引入时态偏序关系,由此就可以引入时态 XML 索引模型概念.

定义 5(时态 XML 索引 TempPartialIndex). 时态 XML 索引是一个三元组的集合:

$$TempPartialIndex(T_0)=\{\langle TagName,Tagtype,\{[Flag],SPVList\}\rangle\}.$$

(1) *TagName*:语义标签;

(2) *TagType*:语义标签对应的节点类型.共有两类:元素类型,标记为 *E*;属性值类型,标记为 *V*;

(3) {[*Flag*],*SPVList*}:语义标签 *TagName* 上等价类划分的集合,是具有同一语义标签 *TagName* 的节点按照[*Flag*]划分的的所有等价类集合,每个语义标签 *TagName* 的[*Flag*]按照由小至大的顺序排列,[*Now*]的顺序值最大.每个等价类由两部分构成:等价类的 *key* 为[*Flag*],等价类的 *Value* 为一个 *List*.即:*SPVList*.*SPVList* 是语义标签 *TagName* 中属于等价类[*Flag*]的节点描述的列表,每个等价类中节点描述的列表按照节点的有效时间的开始时间的偏序关系排列.如果是元素类型,则每个节点描述为 $start^{profix}$,*start* 为节点有效时间的开始时间,*profix* 是节点的前缀编码;如果节点是值类型,则每个节点描述为 $start^{profix,value}$,*start* 为节点有效时间的开始时间,*profix* 是节点的前缀编码,*value* 是节点的值.

例 3(TempPartialIndex 索引模型实例):图 1 中的时态数据对应 Schema 如图 2 所示.

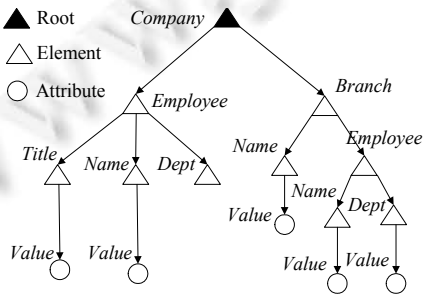

Fig.2 Schema of temporal XML data

图 2 时态 XML 数据的 Schema

相应的{*TagName*}={*company*,*employee*,*name*,*title*,*dept*,*branch*,*name*[*value*],*title*[*value*],*dept*[*value*]}.则基于图 1 的 TempPartialIndex 模型如图 3 所示.由于前缀编码太长,为了方便描述,图中每个节点用一个唯一的数值码来代表前缀编码.

Tagname	TagType	{[*Flag*],*SPVList*}	
		[*Flag*]	*SPVList*
Company	*E*	*Now*	2000^{0}
Employee	*E*	4	2000^{115}
		now	2000^{5},2005^{55}
Name	*E*	4	2000^{100},2000^{120}
		now	2000^{10},2003^{45},2005^{60}
Title	*E*	*now*	2003^{20}
Dept	*E*	4	2000^{130}
		now	2000^{70},2003^{30}
Branch	*E*	4	2000^{95}
		now	2000^{40}
name[*value*]	*V*	4	$2000^{105,beijing}$,$2000^{125,zhanghong}$
		now	$2000^{50,tianjing}$,$2003^{15,yuping}$,$2005^{65,xiaoming}$
title[*value*]	*V*	*now*	$2003^{25,manage}$
Dept[*value*]	*V*	4	$2000^{140,technology}$,$2005^{80,hr}$
		now	$2003^{35,technology}$,$2009^{90,sale}$

Fig.3 Index generated with temporal XML data in figure 1

图 3 根据图 1 的时态 XML 数据所产生的索引

2 TempPartialIndex 数据操作

2.1 TempPartialIndex查询算法

时态 XML 查询分为基于 TXPath[9]的时态查询和时态 XML 文档快照查询两种类型.

2.1.1 基于 TXPath 查询

基于 TXPath 查询就是形如 $Q=A[VT(A)]//B[VT(B)]$的查询,需要同时处理语义、时间和结构的约束.在索引的同一个语义标签中,有若干个基于偏序的时态等价类,时态等价类之间按照跨度大小排序,时态等价类内按照节点有效时间的开始时间的偏序排序,此偏序关系实际上体现了时态等价类中各时间区间的包含关系.根据定理 1~定理 3 所描述的性质,要查找在某个给定的时间区间 *VI* 内有效的节点,只需要查找跨度大于等于 *VI* 跨度的时态等价类;在时态等价类中,只要找两个边界点,边界点之间的节点均满足包含关系.通过该数据结构,将基于二维时间区间上的时态操作转化为了基于偏序的时态等价类上的一维操作,实现了一次一集合的查询处理.算法 2 给出了如何在基于偏序的等价类集合上做时态过滤的算法.算法 3 则整合算法 2 和算法 1 来完成整个时态 XML 的查询处理,即:先进行语义和时态的过滤,再进行 XML 的结构连接.

算法 2. 节点的时态过滤算法 *TempNodeFilter*().

功能:对于任意给定的一个时间区间 Qp,$Vend(Qp)<>Now$,在标签 *TagName* 对应的等价类划分{[*Flag*],*SPVList*}中找到所有节点 v,节点的有效区间 $VT(v)$应满足$\theta(Qp,VT(v))$=true

输入:某个语义标签 *TagName* 对应的等价类划分{[*Flag*],*SPVList*},给定的时间区间 Qp,时态区间关系θ,此算法θ为 during;

输出:{*NodeList*}.

计算 $Vspan(Qp)$;找到 *flag* 值大于等于 $Vspan(Qp)$的最小的[$Flag_j$]

For [$Flag_j$]之后的每一个时态等价类[*Flag*]

{在 *SPVList* 列表中找到 $Vstart(v)$值最大的节点 v,且 v 满足 $Vstart(v)<=Vstart(Qp)$;

在 *SPVList* 列表中找到 $Vstart(v)$值最小的节点 w,且 w 满足 $Vstart(w)>Vend(Qp)-Flag-1$;

将 w 至 v 之间(包括 w 和 v)的节点的信息加到{*NodeList*}中;}

Allen 的时态区间关系运算共有 13 种[18],因为篇幅原因,只列出了最常用的时态关系运算 during 的算法,但

本索引结构能支持 Allen 的 13 种时态运算关系.

算法 3. $A[VT(A)]//B[VT(B)]$的时态 XML 查询算法.

功能:找出在 $VT(B)$时间内有效的 B 节点,且其祖先 A 节点在 $VT(A)$时间内有效

输入:$A[VT(A)]//B[VT(B)]$;

输出:满足查询要求的标签为 B 的节点列表 *ResultList*.

Step 1. 分别找到语义标签为 A 和 B 的所有等价类划分,调用算法 2 得到分别满足时态关系 $VT(A)$和$VT(B)$的节点集[*SPVListA*]和[*SPVListB*];

Step 2. 调用算法 1,找到[*SPVListB*]中其祖先在节点集[*SPVListA*]中的所有 B 节点.

2.1.2 快照查询

快照查询是由时态 XML 图确定的时刻 T_0 的子图,可看作是依据 T_0 对 XML 的切片.

对一个时态 XML 文档在时刻 T_0 进行快照查询,可以看做基于 TXPath 查询的一个特例.

将快照时刻点 T_0 看做时间区间 $VT=[T_0,T_0]$的 TXPath 查询,其余部分可以参照算法 1 和算法 2,区别仅在于:此时,算法 1 的 Step 2 中 $Flag=0(Vspan(VT)=0=T_0-T_0)$.

2.2 TempPartialIndex的更新

由于时态XML数据记录的是对象的历史,当一个对象在现实世界失效,只是将对象有效时间的结束时间修改为失效的时间点,不存在节点的删除,所以节点的删除等效于节点有效时间的修改.所以本文只讨论时态 XML 数据节点的插入和修改对索引 TempPartialIndex 的影响,同时,只讨论涉及到时态属性变更的插入和修改操作.

算法 4. 节点插入算法.

功能:在时态 XML 索引中加入新节点

输入:父节点的查询语句,插入的节点的语义标签、有效时间区间 *VI*;

输出:插入操作是否成功.

1. 根据算法 3 查找到要插入的父节点,新插入的节点总是父节点的最后一个孩子,假设为其父节点的第 i 个孩子;
2. 为该节点分配前缀编码,前缀编码=父节点的前缀编码.i;
3. 根据插入节点的语义标签,找到索引模型中该语义标签的等价类划分集合;
4. 如果 $Vend(VI)=Now$,则把节点加入到等价类[*Now*]中;否则,计算 $Vspan(VI)$,确定要加入的等价类[*Flag*];
5. 在该等价类中查找到小于等于 $Vstart(VI)$的 *start* 值中最大的节点,插在该节点之后.

算法 5. 节点有效时间的修改.

功能:在时态 XML 索引中修改某个节点的有效时间

输入:需要修改的节点的查询语句,修改的有效时间区间 *VI*;

输出:修改操作是否成功.

1. 根据算法 3 查找到要修改节点所在的语义标签的等价类划分集合的位置,删除该节点的节点描述;
2. 如果 $Vend(VI)=Now$,则把节点加入到该等价类划分集合的等价类[*Now*]中;否则,计算 $Vspan(VI)$,确定要加入的等价类[*Flag*];
3. 在该等价类中查找到小于等于 $Vstart(VI)$的 *start* 值中最大的节点,插在该节点之后.

3 时间变量 Now 的处理

本节通过扩展基于时态等价关系的偏序结构,使其支持 *Now* 变量的处理,并能快速地处理时态变量区间的快速查询和更新.

定义 6(时间变量区间的等价关系及等价类偏序结构). 对于时态 XML 文档中所有节点 $v_i(1<=i<=n)$的 $Vend(v_i)=Now$,则 v_i 和 v_j 之间存在着时态等价关系,记为$(v_i)\equiv(v_j)$.节点 $v_1,v_2,\ldots,v_n$ 属于同一个等价类[*Flag*],[*Flag*]

记为[*Now*];在等价类[*Now*]中,所有节点描述的列表按照节点的有效时间的开始时间的偏序关系排列.

定理 5([***Now***]**等价类偏序结构的时态特性**). 对于任意的两个节点 u 和 v,两个节点都属于[*Now*]等价类,则有如下属性:

(1) 如果 $VT(u)\subseteq VT(v)$,则节点 v 在偏序结构中的位置先于节点 u;

(2) 对于任意的一个有效时间区间 *VI*,在等价类[*Now*]中,如果 $Vspan(VI)\neq Now$,则在等价类[*Now*]中不存在一个节点 u,使得如下的时态关系成立:$Equal(VI,VT(u))$=true,$Contains(VI,VT(u))$=true,$After(VI,VT(u))$=true,$Overlapped\text{-}by(VI,VT(u))$=true,$Met\text{-}by(VI,VT(u))$=true,$Started\_by(VI,VT(u))$=true,$Finishes(VI,VT(u))$=true,$Finished\_by\ (VI,VT(u))$=true.

基于定义 6 的结构,时态过滤算法中存在 3 种情况会涉及到 *Now* 变量:前两种情况是给定的查询条件的时间区间的结束时间为 *Now*,在非[*Now*]等价类和[*Now*]等价类中分别查找满足 $\theta(Qp,VT(v))$=true 的节点;第 3 种情况是给定的查询条件的时间区间的结束时间不为 *Now*,在[*Now*]等价类中查找分别满足 $\theta(Qp,VT(v))$=true 的节点.以下给出 3 种情况下基于 *Now* 变量的时态过滤算法.

算法 6. 基于 *Now* 的时态过滤算法 *NowTempNodeFilter*.

功能:对于给定的一个时间区间 *Qp*,在标签 *TagName* 对应的等价类划分{[*Flag*],*SPVList*}中找到所有节点 v,节点的有效区间 $VT(v)$应满足 $\theta(Qp,VT(v))$=true

输入:某个语义标签 *TagName* 对应的等价类划分{[*Flag*],*SPVList*}, 给定的时间区间 *Qp*,时态区间关系 θ,此算法 θ为 during;

输出:{*NodeList*}.

对于 *Flag*=*Now* 的时态等价类

{根据定理 5 属性(1),在 *SPVList* 列表中找到 $Vstart(v)$值最大的节点 v,v 满足 $Vstart(v)<=Vstart(Qp)$;

v 以及 v 之前的节点信息加到结果集{*NodeList*}中;}

对于 *Flag*<>*Now* 的时态等价类

根据定理 5 属性(2),If $Vend(Qp)$=*Now* then 找不到任何一个节点满足条件

算法 6 是第 2.1.1 节中算法 2 的特例补充,算法 2 的功能是进行时态关系过滤和匹配运算,但不涉及到 *now* 变量的任何处理,而算法 6 则针对所有涉及到 *now* 变量的情况进行时态关系运算的补充处理.

例 4(包含时态变量的时态关系运算实例):语义标签为 *A* 的基于偏序的时态等价类划分为{([1],2004^{75},2007^{99}),([2],2000^{15},2001^{87}),([5],2000^{115},2001^{78},2002^{178},2003^{194},2005^{233},2008^{245}),([*Now*],2000^{5},2001^{50},2005^{200},2007^{66})},请找出在[2002,2005]内有效的节点.

时间区间[2002,2005]的跨度为 3,目前共有 4 个等价类,找到大于等于 3 的最小的等价类[5](等价类[1]、等价类[2]中不可能有满足条件的节点),等价类[5]的节点跨度大于 3,首先可以用二分法/B 树快速找到小于或等于 2002 的最大 *start* 值的节点 2002^{178},然后找到大于 $Vend(Qp)-Flag$=2005−5−1=1999 的最小 *start* 值的节点 2000^{115},这两个节点之间的节点均满足条件.即{2000^{115},2001^{78},2002^{178}}.

在等价类[*Now*]中,找到小于或等于 2002 的最大 *start* 值的节点 2001^{50},该节点及其之前的节点均满足条件,即{2000^{5},2001^{50}}.

所以,满足时态过滤的节点为

$$\{2000^{115},2001^{78},2002^{178}\}\cup\{2000^{5},2001^{50}\}=\{2000^{115},2001^{78},2002^{178},2000^{5},2001^{50}\}.$$

在快照查询中有一种特殊的快照查询,即当前数据库快照,该查询会在整个 XML 文档记录的数据历史中获得所有对象当前的数据.即,对时态 XML 文档当前的“切片”.这是一类非常普遍的查询.由于所有结束时间为 *Now* 的节点都在同一个等价类中,所以当前快照的查询变得十分的简单,只需要将[*Now*]等价类中所有节点按照前缀编码排序,即可得到按深度遍历的当前快照的切片.

4 仿真与评测

为检验 TempPartialIndex 的基本性能,本文设计了相应的仿真实验.实验硬件环境为 Inter(R) Core(TM)i5-3230M CPU,主频为 2.60GHz,主存容量为 4GB;外存容量为 500GB.实验在 Windows 7 系统平台下进行.

因为索引 TempSumIndex[14]具有最好的时态 XML 查询效率,即,具有最低的时间复杂度,所以本文选择其作为仿真比较对象.

仿真使用的时态 XML 数据节点数从 10 万~45 万个,每次递增 5 万个节点,内部节点平均时间跨度为 500 个单位,叶节点平均时间跨度为 200 个单位,数据由自定义的程序随机生成,但数据均满足时态约束.

4.1 索引构建

由于 TempSumIndex 索引构建时态数据结构的效率的时间复杂度为 $O(n^2)$[14],而且 TempPartialIndex 索引构建时态数据结构的时间复杂度 $O(n\log n)$,而且在构建索引模型时,由于 TempSumIndex 索引是根据 1-index 路径摘要来进行分类的,所以比 TempPartialIndex 按照语义标签分类的搜索过程要复杂.

索引构建性能评测结果如图 4 所示.

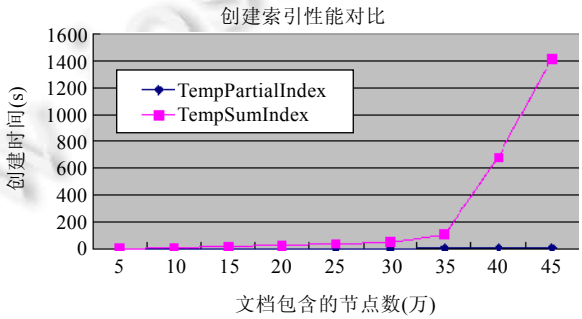

Fig.4 Performance evaluation of building index

图 4 构建索引性能评测

由图 4 也可以看出,语义协同时态 XML 索引 TempPartialIndex 的构建时间开销要远远小于 TempSumIndex 索引构建时间开销,这进一步验证了 TempPartialIndex 索引在进行索引构建时具有比较好的时间复杂度,体现了 TempPartialIndex 的优越性.

4.2 查询功能

4.2.1 TXPath 查询

实验采用的时态约束查询分为:

(1) 无时态约束类型:$Q_1(A//B)$;

(2) 时态约束类型:$Q_{2\text{-}1}(//A[VT(A)])$,$Q_{2\text{-}2}(A//B[VT(B)])$,$Q_{2\text{-}3}(A[VT(A)]//B)$,$Q_{2\text{-}4}(//A[VT(A)]//B[VT(B)])$.

$Q_1(A//B)$是不涉及时态的相对路径查询,其查询处理由语义查询器和结构查询器协同完成.在经过语义过滤后,如果子孙层有 Q 个节点,祖先层有 W 个节点,则祖先/子孙匹配效率会非常慢,本文采用了前缀编码结合 hash 映射的方法,取代了对“值”的缓慢搜索.通过时间复杂度为 $O(1)$,就可以将某个查询值对应的节点对象找出.由于子孙层有 Q 个节点,通过子孙节点的前缀编码得到祖先节点的前缀编码,再通过 hash 映射判断祖先层中是否存在着此前缀编码对应的节点,所以算法的效率为 $O(Qh)$;TempSumIndex 采用前序编码的方式判断祖先/子孙关系,匹配算法的效率为 $O(Q\log Q)$.因而,TempPartialIndex 在处理 $Q_1(A//B)$查询时具有优越性.性能对比如图 5 所示.

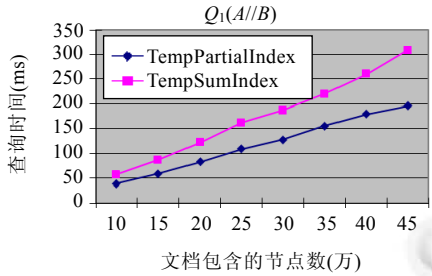

Fig.5 Performance evaluation of query $Q_1(A//B)$

图 5 $Q_1(A//B)$查询性能评测

$Q_{2\text{-}1}(//A[VT(A)])$,$Q_{2\text{-}2}(A//B[VT(B)])$,$Q_{2\text{-}3}(A[VT]//B)$,$Q_{2\text{-}4}(//A[VT(A)]//B[VT(B)])$均为带时态约束的查询类型,这类具有时态约束的查询整个查询过程分为 3 个部分:语义查询处理、时态查询处理、结构查询处理.通过第 1 类查询的比较,TempPartialIndex 的语义部分和结构部分的处理都具有一定的优势,在时态关系运算方面,TempSumIndex 时态运算的复杂度主要是 $O(mb\log n)$,m 为需要处理的时态摘要节点的个数,b 为每一个时态摘要节点的线序划分包含的线序分枝的个数,n 为其时态基本数据结构 LOB 中的元素个数;而 TempPartialIndex 时态运算的复杂度主要是 $O(kn\log m)$,k 是语义节点的个数,n 是需要监测的等价类数,m 是等价类中节点的个数.由于 TempSumIndex 的线序划分需要时间区间之间满足嵌套的关系,及其进行划分的条件要更严苛,所以线序划分的个数会更多.而且 TempPartialIndex 的时态运算对于跨度小于自己的等价类是不需要检查的,而 TempSumIndex 需要检查每一个线序分支.对于查询时间区间的结束时间是 *Now* 的时态运算,TempPartialIndex 只需要检查一个 *Flag=Now* 的时间区间,所以运算的效率为 $O(\log m)$,而基于 *Now* 的查询是一种非常普遍的时态查询.

第 2 类的 4 种时态运算的性能比较如图 6~图 9 所示.

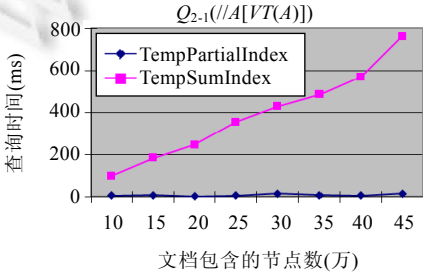

Fig.6 Performance evaluation of query $Q_{2\text{-}1}$

图 6 $Q_{2\text{-}1}$ 查询性能评测

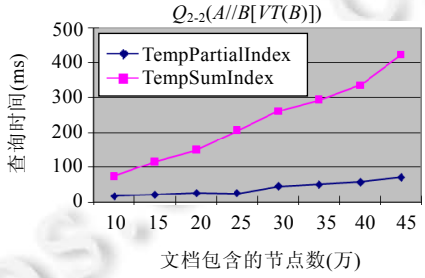

Fig.7 Performance evaluation of query $Q_{2\text{-}2}$

图 7 $Q_{2\text{-}2}$ 查询性能评测

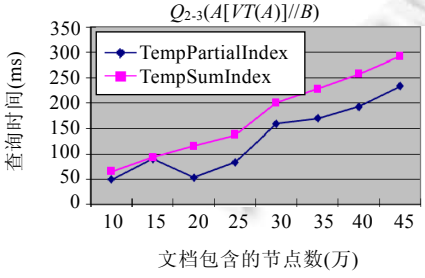

Fig.8 Performance evaluation of query $Q_{2\text{-}3}$

图 8 $Q_{2\text{-}3}$ 查询性能评测

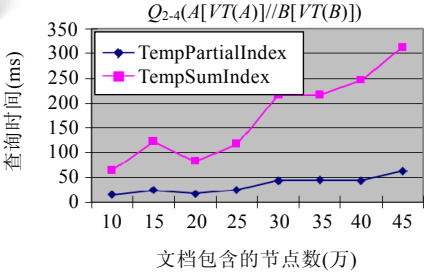

Fig.9 Performance evaluation of query $Q_{2\text{-}4}$

图 9 $Q_{2\text{-}4}$ 查询性能评测

4.2.2 快照查询

时态 XML 数据检索的另一种典型类型就是快照查询,也是时态数据所特有的查询方式.

对于时间点 VT_0=[VT_0,VT_0]的快照查询可以分为两种类型:$Q_{3\text{-}1}(A[VT_0(A)]//B[VT_0(B)])$和 $Q_{3\text{-}2}(VT_0)$两种类型.

对于 $Q_{3\text{-}1}(A[VT_0]//B[VT_0])$快照查询可以归结为是 TXPath 中 $Q_{2\text{-}4}(A[VT(A)]//B[VT(B)])$的一种特殊情形,一方面,$A$ 和 B 的时态约束相同;另一面,时态运算的关系不再是 during,而是 equal.由于 TempSumIndex 需要检查每一个线序分支,因而算法的复杂度为 $O(mb\log n)$;但索引 TempPartialIndex 只需要检查时间区间跨度等于 $VT(Qp)$跨度的时态等价类,满足条件的等价类最多只有 1 个,在此等价类中,只需要查询 VT_0 所在的位置,所以运算的效率为 $O(\log m)$,因而 TempPartialIndex 的效率较优.两种算法的性能对比如图 10 所示.$Q_{3\text{-}2}(VT_0)$最典型的查询就是当前快照查询,即,基于 *now* 的快照查询,该查询的性能分析详见第 4.2.4 节.

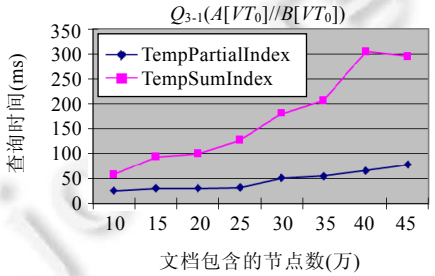

Fig.10 Performance evaluation of query $Q_{3\text{-}1}(A[VT_0]//B[VT_0])$

图 10 $Q_{3\text{-}1}(A[VT_0]//B[VT_0])$查询性能评测

4.2.3 时态 XML 更新

插入的对比实验有两组,第 1 组对比是在 50000~450000 节点的数据中插入 50 个节点(如图 11 所示),第 2 组对比实验实在 10 万节点的 XML 文档中分别插入不同的节点个数,节点的个数按照 10 递增(如图 12 所示).由图 11、图 12 可以看出,TempPartialIndex 要优于 TempSumIndex.由于 TempSumIndex 索引在插入和修改的过程都可能会引起其时态数据结构线序划分中相关线序分枝的分裂和新线序分枝的建立,而 TempPartialIndex 只需在其应该所在的等价类中插入即可,不会导致任何等价类的分裂和重建,所以其时间复杂度等效于时态关系运算的效率.而节点有效时间的修改首先要找到节点原来有效时间所在的等价类,进行删除,然后在新的等价类中插入.其时间复杂度为两倍时态关系运算的效率.

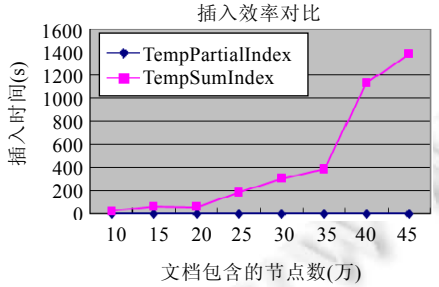

Fig.11 Performance evaluation of inserting (1)

图 11 插入性能评测(1)

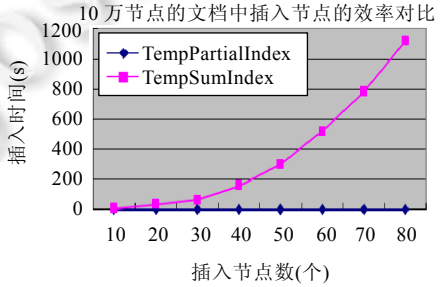

Fig.12 Performance evaluation of inserting (2)

图 12 插入性能评测(2)

4.2.4 基于 *Now* 变量的查询

由于索引 TempPartialIndex 中所有结束时间为 *Now* 变量的节点都在一个时态等价类中,这使得 *Now* 变量的时态过滤只涉及到一个等价类,等价类中的偏序结构使得只需要找到此等价类中查询范围的起点,所以运算的效率为 $O(\log m)$(m 为文档中结束时间为 *Now* 的节点总数).而 TempSumIndex 索引需要检查每一个线序分支

中符合时态约束的节点,因而算法的复杂度为 $O(mb\log n)$(m 为文档中结束时间为 *Now* 的节点总数,n 为 TempSumIndex 索引中线序分支的总数),所以 TempPartialIndex 的效率较优,性能对比如图 13 所示.而当前快照查询也是类似,对于索引 TempPartialIndex 是将时态等价类[*Now*]中所有节点取出,而 TempSumIndex 索引需要检查每一个线序分支.性能对比如图 14 所示.

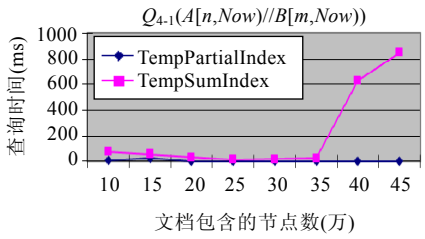

Fig.13 Performance comparison of query $Q_{4\text{-}1}$

图 13 $Q_{4\text{-}1}$ 查询性能对比

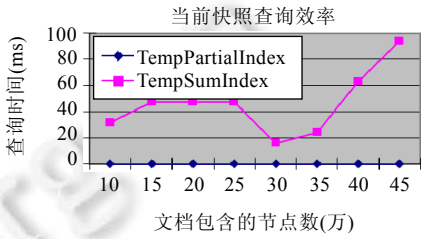

Fig.14 Query time comparison of current snapshot

图 14 当前快照的查询时间对比

5 结 语

时态 XML 的索引研究是提高时态 XML 数据查询效率的关键技术.时态 XML 索引通常将时态约束作为一种特殊的值约束处理,查询过程中,时态约束的处理往往放在最后.事实上,由于时态 XML 数据中每个节点都具有时间标签,数据时间信息的过滤和匹配在整个数据过滤过程中占有比较大的比重,先行进行时间信息过滤可以大大减少获取最终结果过程中的工作量,这类似于关系数据库中先选择投影再连接的查询优化思路.本文借鉴此思想,将基于偏序的时态等价类数据结构整合到非时态的 XML 索引的语义层之中,提出了一种新颖的时态 XML 索引 TempPartialIndex.数据处理过程采用了不同于现有相关工作的方式,即:先处理语义和时态约束,再进行结构的连接.仿真评估表明,TempPartialIndex 索引具有可行性与有效性.

References:

[1] Nørvåg K, Nybo AO. DyST: Dynamic and scalable temporal text indexing. In: Norvag K, ed. Proc. of the Temporal Representation and Reasoning. Budapest: IEEE, 2006. 204−211. [doi: 10.1109/TIME.2006.12]

[2] Bin-Thalab R, El-Tazi N, El-Sharkawi ME. TX-Kw: An effective temporal XML keyword search. Int'l Journal of Advanced Computer Science and Applications. 2013,4(6):217−226.

[3] Faisal S, Sarwar M. Temporal and multi-versioned XML documents: A survey. Information Processing and Management, 2014, 50(1):113−131. [doi: 10.1016/j.ipm.2013.08.003]

[4] Clifford J, Croker A, Grandi F, Tuzhilin A. On temporal grouping. In: Clifford J, ed. Proc. of the Recent Advances in Temporal Databases. Zurich: Springer London, 1995. 194−213. [doi: 10.1007/978-1-4471-3033-8_11]

[5] Kepser S. A simple proof for the turing-completeness of XSLT and XQuery. In: Proc. of the Extreme Markup Languages. 2004.

[6] Fernández M, Siméon J. Growing XQuery. In: Cardelli L, ed. Proc. of the ECOOP 2003—Object-Oriented Programming. Darmstadt: ECOOP, 2003. 405−430. [doi: 10.1007/978-3-540-45070-2_18]

[7] Nørvåg K. The design, implementation, and performance of the v2 temporal document database system. Information and Software Technology, 2004,46(9):557−574. [doi: 10.1016/j.infsof.2003.10.006]

[8] Mandreoli F, Martoglia R, Ronchetti E. Supporting temporal slicing in XML databases. In: Ioannidis Y, Scholl MH, eds. Proc. of the Advances in Database Technology (EDBT 2006). Berlin, Heidelberg: Springer-Verlag, 2006. 295−312. [doi: 10.1007/11687238_20]

[9] Rizzolo F, Vaisman A. Temporal XML: Modeling, indexing, and query processing. The VLDB Journal, 2008,17(5):1179−1212. [doi: 10.1007/s00778-007-0058-x]

[10] Amagasa T, Yoshikawa M, Uemura S. A data model for temporal XML documents. In: Ibrahim M, Küng J, Revell N, ed. Proc. of the Database and Expert Systems Applications. London: Springer-Verlag, 2000. 334−344. [doi: 10.1007/3-540-44469-6_31]

[11] Zheng TK, Wang XJ, Zhou YC. Indexing temporal XML using FIX. In: Liu WY, Luo XF, Wang FL, Lei JS, eds. Proc. of the Web Information Systems and Mining. Shanghai: Springer-Verlag, 2009. 224–231. [doi: 10.1007/978-3-642-05250-7_24]

[12] Zhao L, Wang XJ. Indexing temporal XML using UB-tree. Science of Computer, 2008,35(3):71–72 (in Chinese with English abstract).

[13] Mendelzon AO, Rizzolo F, Vaisman A. Indexing temporal XML documents. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB, ed. Proc. of the 30th Int'l Conf. on Very Large Data Bases—Vol.30. Toronto: VLDB Endowment, 2004. 216–227. [19] Dewey decimal classification. http://www.oclc.org/dewey/

[14] Guo H, Ye XP, Tang Y, Chen LW. Temporal XML index based on temporal encoding and linear order partition. Ruan Jian Xue Bao/Journal of Softwarer, 2012, 23(8):2042–2057 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4161.htm [doi: 10.3724/SP.J.1001.2012.04161]

[15] 叶小平,汤庸,郭欢,陈罗武,朱君,陈铠原.时态索引技术研究及其应用.中国科学(F 辑:信息科学),2009,19(12):1258–1270.

[16] Dyreson CE, Jensen CS, Snodgrass RT. Now in temporal databases. In: Liu L, Özsu MT, ed. Proc. of the Encyclopedia of Database Systems. Springer US, 2009. 1920–1924. [doi: 10.1007/978-0-387-39940-9_248]

[17] Ye XP, Tang Y. Semantics on "Now" and calculus on temporal relations. Ruan Jian Xue Bao/Journal of Softwarer, 2005,16(5): 838–845 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/838.htm

[18] Allen JF. Maintaining knowledge about temporal intervals. Communications of the ACM, 1983,26(11):832–843. [doi: 10.1145/182.358434]

[19] Dewey decimal classification. http://www.oclc.org/dewey/

附中文参考文献:

[12] 赵林,王新军.使用 UB-tree 索引时态 XML.计算机科学,2008,35(3):71–72.

[14] 郭欢,叶小平,汤庸,陈罗武.基于时态编码和线序划分的时态 XML 索引.软件学报,2012,23(8):2042–2057. http://www.jos.org.cn/1000-9825/4161.htm [doi: 10.3724/SP.J.1001.2012.04161]

[17] 叶小平,汤庸.时态变量"Now"语义及相应时态关系运算.软件学报,2005,16(5):838–845. http://www.jos.org.cn/1000-9825/16/838.htm

汤娜(1975－),女,浙江萧山人,博士,副教授,主要研究领域为时空数据库.

叶小平(1955－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为时空数据库.

汤庸(1964－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为协同工作与数据库.

彭鹏(1987－),男,软件设计师,主要研究领域为数据库技术,大数据管理.

杜梦圆(1990－),女,硕士生,主要研究领域为数据库技术,大数据管理.