Virtual eXist-db: Liberating Hierarchical Queries from the Shackles of Access Path Dependence

Curtis E. Dyreson  
Dept. of Computer Science  
Utah State University, USA  
Curtis.Dyreson@usu.edu

Sourav S Bhowmick  
School of Computer Engineering  
Nanyang Technological University, Singapore  
assourav@ntu.edu.sg

Ryan Grapp  
Dept. of Computer Science  
Utah State University, USA  
Ryan.Grapp@aggiemail.usu.edu

ABSTRACT

XQuery programs can be hard to write and port to new data collections because the path expressions in a query are dependent on the hierarchy of the data. We propose to demonstrate a system to liberate query writers from this dependence. A plug-and-play query contains a specification of what data the query needs in order to evaluate. We implemented virtual eXist-db to support plug-and-play XQuery queries. Our system adds a virtualDoc function that lets a programmer sketch the hierarchy needed by the query, which may well be different than what the data has, and logically (not physically) transforms the data (with information loss guarantees) to the hierarchy specified by the virtualDoc. The demonstration will consist of a sequence of XQuery queries using a virtual hierarchy, including queries suggested by the audience. We will also demonstrate a GUI tool to construct a virtual hierarchy.

1. INTRODUCTION

Hierarchical data has existed, in one representation or another, from the dawn of databases to today. About 40 years ago, it was commonplace to store data in the hierarchical data model. Today, hierarchical data in XML, HDF, SGML, or JSON is stored and used in many applications.

Hierarchical data has remained popular in spite of the tight coupling of path expressions with hierarchies. A path expression is specification of the location of a datum in the hierarchy. In his seminal paper on the relational data model, E. F. Codd argued that one problem with the hierarchical model is that path expressions tightly couple queries to physical hierarchies [3]. If the hierarchy changes, a working query may break. Or if the data is slightly different than what the user understands, the query may not work. A path expression that does not match a physical hierarchy will usually evaluate to an empty answer rather than being flagged as an error. For instance a mismatched path expression in Javascript to locate a value in a JSON structure will yield an undefined value rather than throw an exception.

Various strategies have been researched to loosen the tight coupling to improve the portability and ease of writing queries on hierarchical data. Codd changed the data model, but four other distinct solutions have been researched.

1. Rewrite the data - Physically transform the data to the desired hierarchy [5,8,13]. But it can be excessively expensive to transform a data collection, especially when a query uses only a fraction of the data.

2. Rewrite the query - Evaluate the query through a (data transformation) view, c.f. [12, 14]. The chief drawback is that a view is specific to a hierarchy, so each hierarchy needs its own view.

3. Reinterpret the query - Relax or change the query to explore a range of “close” hierarchies [1, 4, 10, 11, 15]. But since query evaluation does not transform the hierarchy, the result is formatted in the source data’s hierarchy.

4. Reinterpret the data - Virtual eXist-db uses this approach. We introduced a numbering system called virtual prefix-based numbering (vPBN) and showed that a vPBN number can be moved to a new location within a hierarchy, yet be used just like a PBN number to determine location-based relationships in the context of the new location [6]. In effect, vPBN virtually rather than physically transforms data and supports query evaluation in the transformed data space. We implemented vPBN in eXist-db, creating virtual eXist-db.

2. THE DEMONSTRATION

We plan to demonstrate virtual eXist-db in a series of steps. At each step we plan to interact with the audience by asking for variations on the queries such as using different virtual hierarchies, re-formulating the query, adding where clause constraints, etc. Users can also try ad hoc queries using our interactive demo at cs.usu.edu/~cdyreson/virtualHierarchies.

Step 1: Query Evaluation in eXist-db: We will start the demonstration by familiarizing the audience with (non-virtual) eXist-db, a native XML DBMS. We plan to use eXist-db’s query sandbox running in a web browser for evaluating the queries on a laptop as shown in Figure 1. The sandbox communicates with an eXist-db server running on the same machine.

Our first query lists for each book its title, publisher, and list numbering (introduced a numbering system called vPBN in eXist-db, creating virtual eXist-db) and showed that a vPBN number can be moved to a new location within a hierarchy, yet be used just like a PBN number to determine location-based relationships in the context of the new location [6]. In effect, vPBN virtually rather than physically transforms data and supports query evaluation in the transformed data space. We implemented vPBN in eXist-db, creating virtual eXist-db.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.

Proceedings of the VLDB Endowment, Vol. 8, No. 12  
Copyright 2015 VLDB Endowment 2150-8097/15/08.
a list of <author>\$a</author>, through a <book>ancestor. When
we run the query on the XML data model instance shown in
Figure 3, the query will produce the data model instance shown
in Figure 4. The query is a kind of data transformation, i.e., it
transforms the data into a new hierarchy. The new hierarchy is given in
the return clause where <author> elements are placed as chil-
dren of <title> elements, as long as the authors are related to
the title through a (least common) <book> ancestor.

Next we give a query to count the number of authors for each
title. To make it easy to count, we embed the data transformation
query as an “inner” query in a nested query as shown in Figure 5.
This is akin to using the data transformation query as a view, and
use a query rewriting technique to combine the query with the view.
We then modify the query to limit the books to those published by
“Addison-Wesley” by adding a where clause as shown in Figure 6.

Finally, we introduce an alternative physical hierarchy for the in-
put data where <publisher> is the parent of <book> as shown
in Figure 7. We rerun the query on this data showing that the count
queries no longer produce a correct answer, i.e., both queries are
tightly-coupled to a hierarchy.

**Step 2: Introducing Virtual eXist-db.** We now demonstrate vir-
tual eXist-db. The demonstration uses the same query sandbox
in a web browser communicating with a virtual eXist-db server.
In terms of functionality, virtual eXist-db is like eXist-db except
that the former supports a virtualDoc() function which can be
used wherever a doc() function is used in an XQuery query. The
doc() function names the document that is to be queried. The
virtualDoc() function has an additional string parameter,
which is a virtual DataGuide (vDataGuide) specification. A
vDataGuide is a structural summary or DataGuide [7] of the de-
sired (virtual) hierarchy for the data. It describes the desired, vir-
tual hierarchy rather than the data’s physical hierarchy. To make the
query to count Addison-Wesley authors per book easier to write, we
want the hierarchy specified by the vDataGuide given below.

title { publisher author { name } }

In the vDataGuide the children of an element type are listed within
braces. The children of <title> elements are <publisher> and
<author>, and and <name> is a child of <author>. The
vDataGuide given above essentially represents the data transfor-
mation view of Figure 2, plus the <publisher> element. The
virtual hierarchy is shown in Figure 8.

A query in virtual eXist-db is logically evaluated with respect to
the vDataGuide. An example is shown in Figure 12. The query
specifies that the count query is to be evaluated on the document
described by the vDataGuide. No data is physically transformed,
only the hierarchy of the data is changed so that nodes appear in
the location they should be after the transformation. The query is
subsequently evaluated in the transformed space.

There are two important parts to evaluating in the transformed
space [6]. First, the relationships between nodes potentially change.
For instance, even if node \$x in the parent of node \$y in the physical
hierarchy, in the virtual hierarchy \$x could be a descendant of \$y.
Second, the value of a node potentially changes. In the physical
hierarchy node \$x may have a \$y child, but in the virtual hierarchy \$y
may be a parent of \$x and hence no longer part of \$x’s value.

In the demonstration we will evaluate the query in Figure 12 on
both of the data instance shown in Figure 3 and Figure 7, where
<publisher> elements have moved to be the parents of <book>
elements to show that it counts correctly. We will also demonstrate
two simple variations of the query to show how the virtual hierar-
chy changes node relationships and values.

We will highlight that a vDataGuide serves two roles. First, it
simplifies the specification of data transformation views. Second,
it virtually transforms different hierarchies to the single hierarchy
desired by the user. This makes a query portable since the query
carries with it a specification of the hierarchy that it needs.

**Step 3: The vDataGuide is a Query Guard.** A further benefit is
that a vDataGuide shares all of the properties of a query guard [5],
<i.e., it functions as a type specification for the query. At this point
in the demo we show how a vDataGuide determines and reports on
potential information loss in constructing the virtual hierarchy.

**Step 4: Virtual eXist-db is efficient.** Next, we will demonstrate that
the virtual transformation happens only for data used in the query,
<i.e., it has an efficiency similar to view transformations (see [6]
for more details). We increase the non-Addison Wesley size of the
document to 1GB and rerun the query with the virtualDoc() to
show that the cost remains flat. We perform a similar query with
a doc() function (cost remains flat) and a nested query with a data
transformation view (cost increases dramatically).

**Step 5: A Peek Under the Hood.** In this part of the demo we look
behind the scenes to see vBPN in action. In order to understand
this part of the demonstration it is necessary to learn a bit about
for $t$ in (...data transformation query...)//title
return
<title>{$t/text()} {count($t/author)} </title>

Figure 5: A query to count the authors for each title

for $t$ in (...data transformation query...)//title
where $t$/publisher = "Addison-Wesley"
return
<title>{$t/text()} {count($t/author)} </title>

Figure 6: A query to count the authors for Addison-Wesley titles

vPBN. A transformation could produce two kinds of changes to the location of a node.

1. **Level change** - The level of a node may change. For example, the <title> Y has moved from level 3 in Figure 3 to level 1 in Figure 4.

2. **Parent change** - A node’s parent may change. For example, the <author> D has switched from the second <book> of Figure 3 to the <title> Y in Figure 4.

vPBN adds a level array to each PBN number. The level array records the tree level of each component in a PBN number. Figure 9 depicts the level array below each PBN in the transformed instance of Figure 4. The leftmost <title> has a level array of [1,1,1] indicating that each component in the PBN number is on level 1. The leftmost <name> has a level array of [1,1,2,3] indicating that the first two components represent the ancestor at level 1, the next at level 2, and the last component is at level 3. The level array together with a PBN number forms a vPBN number.

The vPBNs are used to determine location-based relationships, e.g., is some <title> element a parent of a given <name> element. We instrumented virtual eXist-db to open a window with tabs for the first location-based decision made in a query for each pair of element types. The tab shows the vPBN numbers, the kind of relationship decision, and the outcome.

The second, more intricate part is determining a node’s value in with respect to the virtual hierarchy. eXist-db essentially stores the XML in disk blocks as a string with PBN numbers and some other information interspersed with the original XML. Figure 10 shows a simplified representation. In eXist-db a value index maps PBN numbers a combination of a disk block number and offset within the block to facilitate fast retrieval of specific element values from disk [2].

To demonstrate value construction we instrumented virtual eXist-db to open a tabbed window with one tab for each type of value constructed. Within the tab in a formatted XML snippet with text that does not appear in the transformed value is highlighted in red. This shows that virtual eXist-db is grabs the right values to create the transformed value.

**Step 6: Tool to Construct a vDataGuide**

The final part of the demonstration is a new tool to help construct a vDataGuide. The tool is a drag-and-drop editor for a vDataGuide. The tool has two window panes as shown in Figure 11. In the left pane is the DataGuide for a selected document. The right pane is the vDataGuide. Both panes are editable trees. The user drag nodes from the DataGuide pane to the DataGuide pane to create or extend a vDataGuide. A node dragged to a position above another node is treated as its parent, or dragged below to be a child.

The potential information loss in extending a vDataGuide is indicated by color. The text for the node is colored black if there is no information loss, yellow if there is widening information loss, blue if there is narrowing information loss, and red if information loss is widening and narrowing [5]. Information loss for the entire vDataGuide is also represented at the bottom. When a node is selected in the DataGuide pane, it can be right-clicked to clone, that is, to include its descendants. In the vDataGuide pane, nodes can be selected and deleted. Conversions of elements to attributes and vice-versa is also possible.

The “Generate vDataGuide” button at the top opens a text window with the constructed vDataGuide specification to cut-and-paste into a query.

3. **RELATED WORK**

Previous research has focused on front-end or query language-level solutions to the problem of querying transformed data. Much of this research effort has been devoted to discovering the best way to relax the tight coupling of path expressions in a query to the hierarchy of the data. Approaches include techniques to approximately match a path to a hierarchy, c.f., [1], apply XML search c.f., [4], or systems to relax, reinterpret, or rewrite the path expressions in a query, c.f., [10]. But these approaches do not investigate how to transform the XML values in the data; it is the values in the transformed hierarchy rather than the source hierarchy on which queries in the pipeline should be evaluated.

Research in XML data transformation languages is more relevant [5, 8, 13], but these approaches are inefficient since two passes are needed: one to transform the data, the second to query the transformed data. The most relevant front-end research is to combine an XML query with a view, c.f., [12, 14]. We know of no implementation of query rewriting for eXist-db or any other XML DBMS. Views that transform data are cumbersome to write, and element types constructed in the return clause of a view are distinct from seemingly similar element types referred to in path expressions in a query; they potentially have different values which must be first constructed before being queried. In other words, the view must be (temporarily) materialized and then queried. In contrast, our idea is support queries over data transformation views by manipulating the node numbering system rather than by query rewriting. Note however that virtual hierarchies only construct views that are data transformations (which are a common, important kind of view), query rewriting is still necessary for views, in general.
There are strategies for modifying PBN after an update c.f., [9]. Update renumbering is orthogonal to vPBN. The renumbering physically changes the PBN number for every node in an edit, while vPBN does not change any physical node numbers, instead it logically renumbers the data, re-using the extant physical numbers.

4. SUMMARY
Modern hierarchical data management systems, such as eXist-db, rely on numbering systems like prefix-based numbering to efficiently evaluate queries. But the numbering is rendered obsolete when nodes change location in a data transformation. We demonstrate a system called virtual eXist-db that supports query evaluation in a transformed numbering space. To query data, a user specifies a vDataGuide in a virtualDoc() function. The vDataGuide is a textual representation of a virtual hierarchy for the data. The vDataGuide makes a query portable since it represents the hierarchy the query needs and ensures that the physical hierarchy can be (virtually) transformed to the desired, virtual hierarchy without losing information (or with acceptable information loss). We also demonstrate a GUI tool for constructing vDataGuides.

5. ACKNOWLEDGMENTS
This material is based upon work supported by the National Science Foundation under Grant No. 1144404 entitled “III: EAGER: Aspect-oriented Data Weaving.” Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We wish to thank Shuohao Zhang for his insights in starting this research.

6. REFERENCES