IX-Cubes: Iceberg Cubes for Data Warehousing and OLAP on XML Data

Fianny Ming-fei Jian
The Chinese University of Hong Kong
mfjiang@cse.cuhk.edu.hk

Jian Pei
Simon Fraser University
Canada
jpei@cs.sfu.ca

Ada Wai-chee Fu
The Chinese University of Hong Kong
adafu@cse.cuhk.edu.hk

ABSTRACT

With increasing amount of data being stored in XML format, OLAP queries over these data become important. OLAP queries have been well studied in the relational database systems. However, the evaluation of OLAP queries over XML data is not a trivial extension of the relational solutions, especially when a schema is not available. In this paper, we introduce the IX-cube (Iceberg XML cube) over XML data to tackle the problem. We extend OLAP operations to XML data. We also develop efficient approaches to IX-Cube computation and OLAP query evaluation using IX-cubes.

1. INTRODUCTION

Building data cubes [6] has been well recognized as one of the most important and most essential operations in OLAP (On Line Analytical processing). Many methods have been proposed to compute and store data cubes efficiently from relational data, such as [4, 12, 13, 8].

With more and more data stored in XML format, it is natural to extend OLAP to semi-structured data. For example, an OLAP query on the DBLP database [1] may ask for the number of distinct authors grouped by conference, year, publisher, organization (e.g., ACM and IEEE), and their combinations. Online answering such OLAP queries on XML data is far from trivial. At least two challenges may arise in general.

First, how can we handle *incomplete* and *irregular* XML data? Since XML data is semi-structured, and a user-defined schema may not exist, so data entries of the same type may be stored in very different ways. For example, as shown in Figure 1, entries about two books may be represented differently. Moreover, missing data can be common. For instance, some conferences may not have the information about organizations. Second, how can we compute aggregates ?In this paper, we tackle the problem of supporting OLAP on XML data and develop an IX-Cube approach: we construct an iceberg data cube on an XML data set so that various aggregate queries can be answered efficiently. We make the following contributions.

Firstly, *we extend data cubes from relational data to XML data*. The extension is non-trivial. We propose the concept of *IX-Cube* (for Iceberg XML Cubes) and develop methods to handle incomplete and irregular XML data.

Secondly, *we investigate how to use IX-Cubes to answer OLAP queries*. We study both aggregate cell queries and sub-cube queries. To accelerate query answering, we present a B+-tree index on IX-cubes.

To the best of our knowledge, this is the first systematic study on constructing, storing, indexing and using data cubes on XML data.

2. RELATED WORKS

While the study of query processing on semi-structured or XML data has received much attention, there has been little work on OLAP queries on such data. [5] analyzes the opportunities and challenges of analytical processing of XML data. [3, 2] study the problems of extending XQuery to support analytical queries over XML data. [7] explores the problem of integrating XML data and relational data into a "virtual" OLAP DB, over which the OLAP operations are executed. [10] federates the external XML data with OLAP data, and proposes query optimization techniques for such federations. [9] proposes how to compute data cube from distributed data, and use XML to represent the cube and the cube schema. None of these work touches the details of defining, computing and querying a cube computed from XML data. To our knowledge the only work that deals with datacube for XML data is [11]. The semi-structured nature of XML documents is handled by a systematic relaxation process in [11]: the dimensions in a datacube are specified by a target tree and the possibility of missing entries is modeled by relaxing the target tree, forming all possible subtrees by removing branches or replacing parent-child relationship by ancestor relationship. However, this model does not cater for other different possible path patterns for the same dimension. We provide a more general solution by allowing users to specify the desired datacube in a more flexible and more precise manner.

3. PRELIMINARIES: XML AND XPATH

An XML document can be modeled as a tree. We assume that each node in the tree has an attribute **tag**. For an element/attribute node, we take the tag name as the tag. For a text node, the tag is simply "text", and a text node has an additional attribute *textcontent*. Figure 1 shows

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'07, November 6–8, 2007, Lisboa, Portugal.
Copyright 2007 ACM 978-1-59593-803-9/07/0011 ...$5.00.

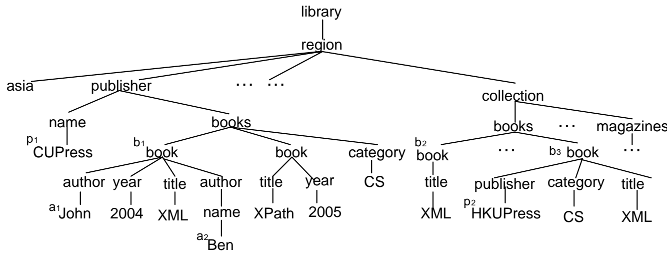

Figure 1: An example XML tree. The labels besides nodes (e.g., p_1 and b_1) are to facilitate some discussion in the paper.

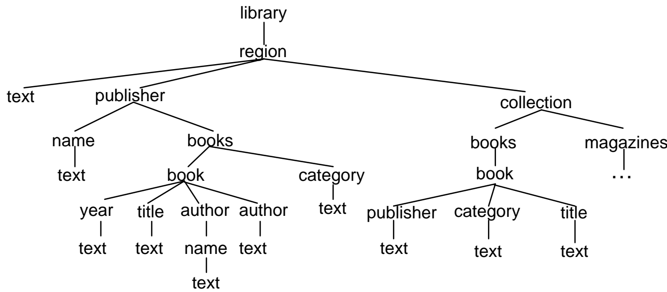

Figure 2: The tag path trie for the example XML tree in Figure 1.

the tree representation of an XML document about literature in a library. Take node b_1 in the figure as an example. $b_1.tag$ =book. We call b_1 a book node. a_1 is a text node, and $a_1.tag$ =“text”, $a_1.textcontent$ =John.

A *tag path* is a list of tags from the root to a leaf node. Clearly, given an XML tree, all distinct tag paths in the tree can be organized into a prefix-trie called the *tag path trie*. For example, Figure 2 shows the tag path trie of the XML tree in Figure 1. Since all text nodes have the same tag “text”, there are typically only a small number of distinct tag paths even in a large XML tree.

Generally, this tag path trie is not a schema. A general schema should be a graph, which leads to tag paths of unknown length. However, for a specific document, there are only a limited number of paths. Therefore, the tag path trie can serve as the hidden schema for the document.

Moreover, an XML document may not conform to any schema, but it must have such a tag path trie describing all the possible distinct tag paths that appear in the document. Therefore, in the following discussion, we assume that such a tag path trie is available by some preprocessing. Apparently, building such a tag path trie needs at most one scan of the XML tree.

XPath is a language for selecting nodes in an XML tree. For example, the XPath expression “/library/region[asia]//book” selects all books nodes which are descendants of a region node with an asia child. An XPath expression can be very complicated. To keep our discussion simple, we consider only XPath expressions using branchings (“[]”), self axis(“.”), child axis(“/”), attribute axis(“@”), descendant axis (“//”), wild card (“*”), and axis for selecting all text children of the context node(“text()”) and parent(“..”). For simplicity, later on we call an XPath expression a **path**.

target entity	//book
measure	$\langle$COUNT,{.},25000 $\rangle$
dimensions	
name	**Paths**
publisher	$\{P_{p_1}$ =./publisher/text(), P_{p_2} =./../../../publisher/@name/text()$\}$
category	$\{P_{c_1}$ =./category/text(), P_{c_2} =./../category/text() $\}$
author	$\{P_{a_1}$ =./author/text(), P_{a_2} =./author/name/text()$\}$

Table 1: An example of IX-Cube specification

4. IX-CUBES

Conceptually, a (relational) data cube [6] is the set of all possible group-by aggregates on a multidimensional data set. For example, given a table (year, author, category, publisher, title), we can specify a data cube using attributes year, category, and publisher as the *dimensions*, using attribute *title* as the *measure*, and using `COUNT()` as the aggregate function. Then, an *aggregate cell* in the cube is an aggregate on a group-by using a subset of the dimensions, such as the total number of books by category and publisher. A data cube is the complete set of all possible aggregate cells.

A data cube can be huge if there are many dimensions and the cardinalities of dimensions are high. Often, users are only interested in significant aggregates. Given a user specified aggregate threshold, an *iceberg cube* is the set of aggregate cells whose aggregates pass the threshold.

4.1 Specifying IX-Cubes

Now, let us consider how to extend data cubes to XML data. Specifying a data cube on XML data is more complicated because of the incompleteness and irregularity of XML, as illustrated in Section 1. Particularly, we need to handle two challenges.

The irregularity. The entries (i.e., the correspondents of tuples in the relational case), the dimension values and the measure attribute values are semi-structured in an XML document. Therefore, we need to specify how the entries, the dimension values, and the measure attribute values can be found and associated with each other.

The incompleteness. Some dimensions may be missing in some entries. Then, we need to provide an effective way to determine whether there are some missing dimension values for an entry, and, if so, how such an entry is aggregated.

We refer to a datacube to be derived from a XML document as an IX-Cube. Table 1 shows an example of an IX-Cube specification on the XML tree in Figure 1. Before we give the formal definition, this example can help us understand some essential issues in the IX-Cube specification.

Specifying an entry is straightforward. We can use an XPath expression to specify the set of target entity set that is of interest for aggregation. For example, the target entity set defined by “//book” indicates that we want to build a data cube about book nodes.

A dimension can also be defined by an XPath expression. However, the path is relative. That is, after an entry is identified, we should evaluate the XPath expression starting

from the entry. The path leads us to the dimension value. For example, in Figure 1, path "./category/text()" may lead to the category of a book, and thus can be used to define dimension category. Due to the irregularity of XML data, we may need more than one path to specify a dimension. For example, in Figure 1, some book entries need path "./category/text()" to find the categories, and some others need path "./../category/text()". In the dimension specification, both paths should be included.

Similarly, we can use a relative path to specify the measure. An aggregate function should be provided. Putting things together, we define IX-Cube, an iceberg cube on XML data, as follows.

DEFINITION 1 (IX-CUBE). Given an XML tree, an **IX-Cube** is specified by a 3-tuple $\langle E_{xcube}, M_{xcube}, D_{xcube}\rangle$, where E_{xcube} is an entity, M_{xcube} is a measure, and D_{xcube} is a set of dimensions. E_{xcube} is given by a path ending in a label of interest. Once we find a node in the XML document that matches E_{xcube}, we call it a **context node**. A **dimension** is defined by 2-tuple $\langle$*DName, Paths*$\rangle$, where *DName* is the dimension name, and *Paths* denote a set of paths whose starting point is the context node. A **measure** is defined by a 3-tuple $\langle$*AggFunction, Paths, min_supp*$\rangle$, where *AggFunction* is an aggregate function, *Paths* is a set of paths to be considered for the aggregation whose starting point is the context node, and *min_supp* is the aggregate threshold. Only aggregates passing the threshold are stored in the IX-Cube. □

Let us consider the example of IX-Cube specification in Table 1 again. Dimension "publisher" says that given a book node (current context node), its publisher information may either be found as a text child of a publisher child node, or as the name attribute of a publisher child of its grand-grand-parent. $\langle COUNT$, {.}, 25000$\rangle$ is a measure, which counts the number of books in the document. If each book has an attribute of inventory, which records the number of copies, a possible measure would be $\langle SUM$, {./inventory/text()}, 300000$\rangle$, which calculates the total number of copies for all books.

4.2 Storing IX-Cubes

We use a tree to store an IX-Cube. Given an XML tree and an IX-Cube definition $\langle E_{xcube}, M_{xcube}, D_{xcube}\rangle$, the resulting IX-Cube is a directed labeled tree $T_{xcube} = \langle root, E, V, l_E\rangle$, where

- *root* is the root of T_{xcube}. V is the set of nodes, and E is the set of edges in T_{xcube}.
- Each $e \in V \cup \{root\}$ has three attributes, **val**, **ins** and **agg**. Here, **val** is a valid dimension value or Φ; $root.val = \Phi$; **ins** is a set of context nodes; and **agg** is the aggregate computed on the measure values of context nodes in **ins**.
- l_E is the labeling function for edges in T, mapping an edge to a dimension name.

Figure 3 shows the IX-Cube computed on the XML tree in Figure 1 given the specification in Table 1. We use the dimension value to represent a node. The number in bracket beside the value is the aggregate. Limited by space, we do not show the context nodes associating with each node, i.e. the **ins** attribute of a node. For example, the **ins** attribute of the node in the dashed circle is $\{b_1\}$. The purpose of keeping such information in the tree is for easy incremental update of the IX-Cube. A path from the root to a node represents a **cell** in an IX-Cube. For example, the node in the dashed circle represents a cell "({CUPress}, *, {Ben,John}:1)", which means the number of books published by CUPress and written by Ben and John together is 1, counting all the categories. Note that a **total order** on the dimensions is assumed, which can be an arbitrary choice or one based on the statistics of queries. In the example, the order is publisher ≺ category ≺ author. In the next section we shall see how the order can affect the performance of answering some queries.

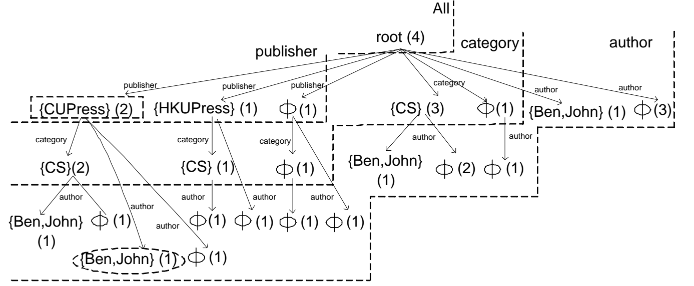

Figure 3: An example of IX-Cube.

As we can see, common prefixes are shared in the tree structure so that space can be saved. Besides, this construct allows the aggregates of the internal nodes to be computed so that the early pruning can be applied based on the iceberg threshold [12].

The dashed lines partition the aggregates for different dimensions. They also separate an IX-Cube into different **layers**, one for each dimension (or concept hierarchy for a dimension). There is a special layer for "All", which is called the *top layer.* The layer containing the leaf nodes is called the *bottom layer.*

5. ANSWERING OLAP QUERIES USING IX-CUBES

In this section, we discuss how an IX-Cube can be used to answer OLAP queries, and how an index can be built to accelerate such queries. Here, we assume that an IX-Cube is materialized. The computation of IX-Cubes will be discussed in Section 6.

5.1 Cell Queries

A **cell query** asks for the aggregate of a specific group-by. For example, one may query "the total number of books written by Ben and John published by CUPress". This query has two dimensions, "publisher" and "author". For convenience, we use an SQL-like syntax as follows.

```
SELECT  COUNT("/library//book/")
FROM    ACUBE(Entity="//book")
WHERE   publisher=CUPress AND
        author=Ben,John
```

The answer to this query is a cell in the IX-Cube, i.e., the cell represented by the node with dashed circle in Figure 3.

Answering a cell query using an IX-Cube is straightforward. To answer the above query, we traverse the IX-Cube starting from the root until we get a path "/root/{CUPress}/{Ben,John}", where the labels l_E(root, {CUPree}) = publisher, l_E({CUPress}, {Ben,John}) = author. Then, the path, together with the aggregate stored in the node with value {Ben,John}, form the resulting cell, i.e. "({CUPress}, *, {Ben,John}:1)".

In general, given a cell query using a set of dimensions $D = \{D_1 = V_1, D_2 = V_2, \ldots, D_m = V_m\}$, where $D_1, D_2, \ldots, D_k$ are in the IX-Cube, we get the path P_{cell} ="/root/e_1/ . . . /e_k" in the IX-Cube such that $e_i.val = V_i$ and $l_E(e_{i-1}, e_i)$ is $D_i.DName$, where $1 \leq i \leq k$ and $e_0 = root$. The cell formed by P_{cell} and $e_k.agg$ should be returned.

5.2 Sub-cube Queries

OLAP queries often involve report-style queries, which are queries on a sub-cube of the iceberg cube. Since the IX-Cube materializes all **cuboids** (group-by's of all subsets of dimensions), we can generate the result from an IX-Cube by looking up the corresponding layers of the IX-cube, e.g., from the root, get all publisher child nodes, and from each such node get all the author child nodes, those nodes form the cuboid. In OLAP, roll-up is to climb a concept hierarchy for a dimension or to reduce the number of dimensions, while drill-down is the opposite operation. By going up or down the branches in the IX-cube, we can also perform roll-up or drill-down operations.

5.3 CubeIndex: a B+-tree Index on IX-Cubes

To accelerate query answering, we propose a B+-tree style index on IX-Cube. We call the index **CubeIndex**. Given an IX-Cube $T = \langle root, E, V, l_E \rangle$, for each node e_k with incoming path "root/e_1/e_2/. . ./e_{k-1}", where $l_E(e_{i-1}, e_i) = d_i (1 \leq i \leq k, e_0 = root)$, we insert e_k into the CubeIndex using the key $\langle d_1, d_2, \ldots, d_k, e_1.val, e_2.val, \ldots, e_k.val \rangle$. We also store $e_k.ins$ and $e_k.agg$.

To answer a cell query, we can search the CubeIndex with dimension names and values as the key. Using the IX-Cube in Figure 3 as an example. To access the cell in dashed circle, we search the CubeIndex by key $\langle publisher, author, \{CUPress\}, \{Ben, John\} \rangle$, and get the aggregate, i.e., 1. Then, a cell "({CUPress}, *, {Ben,John}:1)" is formed and returned.

6. COMPUTING AN IX-CUBE

In this section, we discuss how to compute an IX-Cube. Our proposed cubing algorithm is a bottom-up approach. The input consists of an XML tree T, an IX-Cube definition, $\langle E_{xcube}, M_{xcube}, D_{xcube} \rangle$, where $D_{xcube} = \{D_1, D_2, \ldots, D_k\}$, and each D_i is a 2-tuple $\langle D_i.DName, D_i.Paths \rangle$, and two user-defined parameters, δ and *minsupp*, where δ is for getting valid dimension and measure values, and *minsupp* is for computing iceberg IX-Cube. $T_{xcube} = \langle root, V, E, l_E \rangle$ can be computed by the following steps.

For those dimensions whose values are missing, we use Φ as the place holder. Then, for each context node, its measure value and all dimension values form a tuple. Then, we apply a similar idea as the BUC algorithm [4] to compute the IX-Cube. We store the results into the B+-tree CubeIndex for the resulting IX-cube. We call our algorithm **RCubeAlg** and there are two steps:

Step 1: Flatten the tree. The task of this step is to transform the relevant datacube information from the XML data into a relational table. We call such a table a **base table**. The basic idea is as follows. We scan the XML tree in preorder. For each context node, we pick up its dimension or measure values according to the paths in the dimension or measure definitions. For those dimensions whose values are missing, we using Φ as the place holder. Then all the values form a tuple.

Step 2: Given the base table, we apply the BUC algorithm[4] to compute a cube based on the base table. The output cube is stored as an IX-Cube as introduced in Section 4.2. Note that, as the structure features are eliminated after flattening the tree, the labels of the edges in an IX-Cube computed by RCubeAlgo are dimension names.

7. CONCLUSION

In this paper, we propose a new data structure, IX-Cube, over XML data. Although the purpose of computing data cube over XML data is the same as that of relational data, the technique is quite different due to the difficulties in handling the semi-structure nature of XML data. We first introduce the definition of IX-Cube, which includes the definition of a target entity, a measure and multiple dimensions. Next a computing algorithm is proposed, and OLAP operations over the IX-Cube are defined.

8. REFERENCES

[1] http://www.informatik.uni-trier.de/ ley/db/.

[2] K. Beyer, D. Chamberlin, L. Colby, F. Ozcan, H. Pirahesh, and Y. Xu. Extending XQuery for analytics. In *SIGMOD*, 2005.

[3] K. Beyer, R. Cochrane, L. Colby, F. Ozcan, and H. Pirahesh. Xquery for analytics: Chanllenges and requirements. In *First International Workshop on XQuery Implementation, Experience and Perspective*, 2004.

[4] K. Beyer and R. Ramaktishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD*, 1999.

[5] R. Bordawekar and C. Lang. Analytical processing of xml documents: Opportunities and challenges. In *SIGMOD Record*, volume 34, June 2005.

[6] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *Proc. 1996 Int. Conf. Data Engineering (ICDE'96)*, pages 152–159, New Orleans, Louisiana, Feb. 1996.

[7] M. Jensen, T.Moller, and T. Pedersen. Specifying olap cubes on xml data. In *Journal of Intelligent Information System*, 2001.

[8] L. Lakshmanan, J. Pei, and Y. Zhao. Qc-trees: An efficient summary structure for semantic olap. In *SIGMOD*, 2003.

[9] T. Niemi and M. Niinimaki. Constructing an OLAP cube from distributed XML data. In *DOLAP*, 2002.

[10] D. Pedersen, K. Riis, and T. Pedersen. Query iptimization for olap-xml federations. In *DOLAP*, 2002.

[11] N. Wiwatwattana, H. Jagadish, L. Lakshmanan, and D. Srivastava. X$\hat{3}$: A cube operator for xml olap. In *Proc. 2007 Int. Conf. Data Engineering (ICDE'07)*, 2007.

[12] D. Xin, J. Han, X. Li, and B. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *VLDB*, 2003.

[13] D. Xin, Z. Shao, J. Han, and H. Liu. C-cubing: Efficient computation of closed cubes by aggregation-based checking. In *ICDE*, 2006.