XML Query Algera for Cost-based Optimization *

© Maxim Lukichev Dmirty Barashev

University of Saint-Petersburg
lmax@inbox.ru

Abstract

Several requirements for algebra suitable for efficient cost-based optimization are presented. It is shown that known XML algebras do not fully satisfy this requirements. A new algebra to satisfy better the requirements is introduced.

1 Introduction

Continously growing usage of XML data demands for development of powerful query optimization systems. Optimization approaches for XML databases depend on database type. Relational based XML DBMS decompose documents into conventional or special binary relations [1]. XQuery clauses in such systems are translated to queries in SQL-like language and query processors employ traditional relational query optimization techniques. So called native XML DBMSs use their own data storage formats. Query optimizers in the native XML databases are not as efficient as their relational counterparts. Usually they are limited with some logical optimization methods and follow a naive physical plan. However, an efficient optimizer should consider many equivalent physical plans and choose the best one using some cost function. Execution of the best physical plan may be significantly, sometimes several orders of magnitude faster comparing to naive one. So the problem of generating the optimization space is very important. Optimization space is a set of equivalent expressions in some query algebra and thus the final plan quality is defined by some properties of the chosen algebra. Unlike relational systems, there is no standard algebra for XML queries, but there are many different algebras [12, 14, 4, 13, 8] each having its own pros and cons.

In this work we gather requirements for query algebra suitable for efficient cost-based optimization and propose an algebra based on elements of XAT [14] and Xtasy [12] algebras and which meets the requirements.

2 Related work

Many researches in XQuery optimization are focused on logical optimizations. A number of logical transformations were considered in [9], including semantical optimization, pushing predicates, joins reordering, eliminating redundant document-order sorts. Rules of transforming XQuery queries to forms more suitable for translation to SQL were described in [7].

XML Query Algebra [3] comes from the activity of the W3C XML Query Working Group. That algebra is mainly intended for the formal definition of query languages semantics. The algebra itself is an abstract version of XQuery, where high-level operators (e.g., n-ary for and sortby clauses) are mapped into low-level algebraic operators. Rewriting rules are provided resembling functional programming languages rules and nested relational rules.

Xtasy [12, 5] algebra also as YAT [13], XAT [14] and SAL [4] algebras has operators defined on relational-like structures. Operations similar to relational as selection, projection, join, cross product, order by etc. have appeared in these algebras. But also operations specific to XPath and XQuery have appeared. So in Xtasy they are presented by path and return operations, those similar to bind and tree in YAT algebra. In XAT and SAL algebras for variable binding map operation is used.

TAX [8] is a query algebra developed in the context of the TIMBER project. TAX data model is based on unordered collections of ordered data trees, and each TAX operator takes as input collections of data trees, and produces as output collections of data trees. Unlike YAT and SAL, TAX directly manipulates trees without the need for an explicit intermediate structure. Data extraction and binding are performed by using pattern trees: pattern trees, which resemble Xtasy input filters, describe the structure of the desired data, and impose conditions on them.

MonetDB system [1] worth special attention. Data in that system are stored as binary relations. Queries are translated to special intermediate SQL-like representation and then are executed as SQL. Benchmarks show a significant superiority of MonetDB over native XML DBMS, mostly when working with big documents. The reason is that native XML databases are lacking powerful query optimizers and efficient indexing structures.

3 Analysis of Algebra requirements

Time required to evaluate some query could differ very much depending on the chosen evaluation plan. The main optimizer role is to find the most effec-

* This work was partially supported by RFBR (grant 07-07-00268a).

tive one. The space of available physical plans is mainly defined by properties of algebra operations. And the wider is that space the more effective evaluation plan could be found. Such properties of algebraic operations as commutativity, associativity and idempotance are desirable properties because they extend search space, increasing optimizer's freedom for choosing optimal plan.

One of the most important and widely used constructions of XQuery are nested for-clauses. The order of their evaluation in evaluation plan is significant for performance as order of join operations in relational algebra. Therefore the representation of nested for-clauses with operations with good algebraic properties is an important requirement.

It is impossible to choose an order of operations evaluation without estimation of a size of an intermediate result, for example result of joining two sequences from the given three. Therefore data structures in query algebra must be good enough to represent measurable intermediate results.

XPath expressions also play an important role in XQuery. There may be different plans of evaluating the same XPath expression and some plans may be much more expensive than others [10]. So if XPath would be presented with operations with good algebraic properties probably the more effective plan could be found. It is the last important requirement.

Concluding, XML query algebra suitable for cost-based optimization is expected to satisfy the following requirements:

1. operations are defined on data structures suitable for representing measurable intermediate results
2. nested for-clauses are mapped to operations with good algebraic properties such as commutativity and associativity
3. xpath expressions are also represented by operations with good algebraic properties.

W3C algebra operations defined on sequences. This definition leads to several problems with an intermediate result representation, because sequences do not provide any information about corresponding bound variables and e.t.c. Therefore there are problems with intermediate result storing and estimation that leads to problems with reordering of some expensive operations. So this algebra does not satisfy requirement (1). But this requirement is satisfied by YAT, XAT, Xtasy algebras. These algebras have operations defined on relational-like structures, which are as relations consist of tuples. This structure is suitable for representing intermediate result of joining group of sequences. In XAT algebra such structure called XAT-table, in Xtasy – Env.

The XML Query algebra is very useful for implementing simple XML query processors with ability of logical optimization, but it appears unlikely that it will form the basis for effective implementations of XML query processors with cost-based optimization.

YAT, XAT and Xtasy algebras have representation of nested for-clauses with join operations. These operations have enough good algebraic properties. So the requirement (2) is satisfied by these algebras. But requirement (3) does not completely satisfied by them. The reason is that their operations used for xpath representation do not have complete number of desirable algebraic properties. Therefore it is impossible to arbitrary change evaluation order of operations for navigational expressions.

4 XAnswer Algebra

In this section we propose new algebra called XAnswer. This algebra is based on some elements of XTasy and XAT algebras and satisfy requirements described in previous section. First, it would be described data structures, algebra operations defined on. After that main algebra operations, that are similar to relational, would be introduced. And at last specific for XQuery algebraic operations would be defined and compared with analogues of Xtasy algebra.

4.1 XAnswer data structure

XAnswer algebraic operations are defined on relational-like data structures like in [12, 14]. Below, this structure will be called Envelop. It is represented with a table and consists of tuples which contains XML-node values. Order of table attributes or tuples is not significant.

In case of XQuery each attribute of Envelop is a name of variable that was bound in corresponding subexpression. In case of XPath there are no any variables, therefore some unique identifier is used as corresponindg attribute instead of variable name.

Lets consider a path expression: book/author/address/country. Lets Assume that identifier $\$V_A$ denotes values of nodes with tag name book, identifier $\$V_B$ – author, $\$V_C$ – address, $\$V_D$ – country. The Envelop, obtained after evaluation of considered path expression could be represented with a table shown in Figure 1.

$V_A	$V_B	$V_C	$V_D
a1	*b1*	*c1*	*d1*
a2	*b2*	*c2*	*d2*
an	*bn*	*cn*	*dn*

Figure 1: Envelop structure

XML data model assumes ordering of tags which is missing in our Envelop structure. However for optimization purposes better algebraic properties are more important than ordering so we assume that query result should be additionally sorted if needed. Assuming these two Envelops are equal independent of tuple or attribute order.

4.2 Algebraic operations

Path expressions are main building blocks of XQuery expressions. Also their evaluation is one of the most

expensive elements in XQuery evaluation. Therefore it is important to increase quality of xpath evaluation plans. So query algebra has to provide a wide space of equivalent plans for navigational expressions. This problem could be solved by representation of XPath with operations with good algebraic properties. So XAnswer algebra use structural-join (binary) and data extraction (unary) operations to represent path expressions and each step of path expression is represented with these operations.

4.2.1 Structural-join operation

Structural-join operation appears in many works around the XPath optimization, for example [6, 15, 2]. XAnswer also has this operation with following definition:

$$\text{A} \otimes_{axis_{A_iB_j}} B = \{(x,y) \mid \text{x} \in A,\ y \in B,\ axis_{A_iB_j}(x,y) = true\}$$

Here A, B – two input Envelops. A_i, B_j – attribute identifiers by which join is performed. $axis_{A_iB_j}$ – is a predicate by which join is performed. It returns true, if elements of x and y, corresponding to identifiers A_i, B_j are in axis relation (for example child or parent).

Example 1

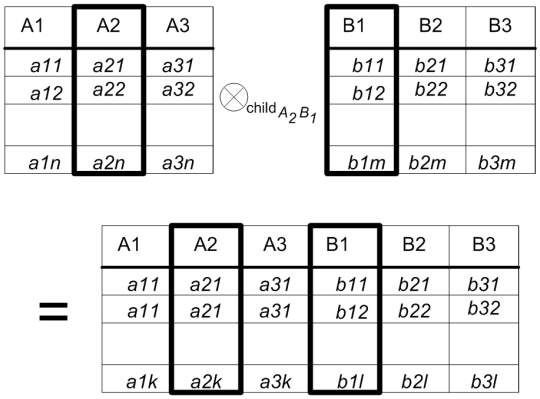

Figure 2: Structural join example

This example shows structural join of two Envelops by child axis for attributes $A2$ and $B1$. Result of joining is a new Envelop, every tuple of which obtained by union of tuples of first and second Envelops in case when tuple element corresponding to attribute $B1$ of the second Envelop is a child of tuple element corresponding to attribute $A2$ of the first Envelop.

Structural join operation has associativity(1) and commutativity(2) properties. These properties could be prooved using operation definition.

$$(1) : (A \otimes_{axis_{A_iB_j}} B) \otimes_{axis_{B_mC_k}} C = A \otimes_{axis_{A_iB_j}} (B \otimes_{axis_{B_mC_k}} C)$$

$$(2) : (A \otimes_{axis_{A_iB_j}} B) = (B \otimes_{axis_{A_iB_j}} A)$$

4.2.2 Data Extraction operations

There are some operations (leaf or unary operations), which produces new Envelop without provided any another as input. These operations are presented with function call and element search operations. GetDocumentRoot operation is a member of function call operations family. This operation generates a new Envelop containing root node for provided document. GetDocuemntRoot operation defined as follows:

$$GetDocumentRoot(Document) = \{x \mid x \text{ is a root element of the } Document\}$$

The next important data extraction operation is GetElement operation. It searches for elements, that satisfy NodeTest condition in the provided set of documents. For example as a NodeTest condition could be a NameTest, i.e. some tag name.

$$GetElement(NodeTest, DocSet) = \{x \mid x \in \text{elements of documents from } DocSet, NodeTest(x) = true\}$$

The result of evaluation of this operation is a new Envelop containing values of nodes satisfying to the NodeTest. As a single attribute of this Envelop, some unique identifier is set. The following example shows XAnswer algebraic expression for some typical XPath twig query.

Example 2

Lets consider following path expression:
document("doc.xml")/manufacturers//dealer/address
The algebraic expression corresponding to this path expression is:

$$((GetDocumentRoot(doc.xml) \otimes_{child_{v_1v_2}} GetElement(manufacturers, doc.xml)) \otimes_{child_{v_2v_3}} GetElement(dealer, doc.xml)) \otimes_{child_{v_3v_4}} GetElement(address, doc.xml)$$

This algebraic expression could be represented with a tree shown in Figure 3. For simplicity in this figure does not provided some information like axis predicates for structural joins. Evaluation is performed by left-deep walking through this tree. It means that for this tree first would be extracted document root node then manufacturers nodes then would be performed structural join by child axis and so on.

Sometimes much more effective plan could be achieved by changing evaluation order of structural joins. So Figure 4. shows alternative equivalent algebraic expression for the expression from Example 2. Equivalence of these expressions could be proved by sequential applying of rule (1). In this case first would be performed joining of manufacturers and document root then joining of dealers and addresses, and at last the structural join by descendant-or-self axis for manufacturers and dealers is performed.

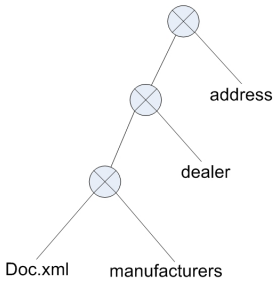

Figure 3: An xpath algebraic expression tree

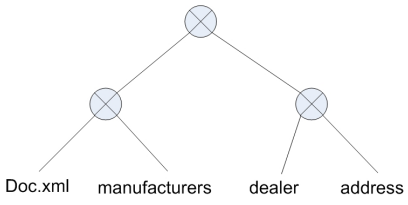

Figure 4: An alternative xpath algebraic expression tree

4.2.3 Classical algebraic operations

Some operations in XAnswer derived from relational algebra. That are such operations as selection, projection, join, cross product, orderby... These operations have the same properties as their relational analogues.
Selection:

$$Select_P A = \{x \in A \mid P(x) = true\}$$

Projection:

$$Project_{A_{i_1}\ldots A_{i_n}} A = \{z \mid \mathrm{z} = (x_{A_{i_1}}, \ldots, x_{A_{i_n}}),\ x \in A\}$$

Join:

$$\mathrm{A} \bowtie_P B = \{(x, y) \mid x \in A,\ y \in B,\ \mathrm{P}(x, y) = true\}$$

4.3 Specific for XQuery operations

For and Let operators also known as variable binding operators are one of the most important operators in XQuery. These operators are similar in that they define a variable in query evaluation context and set to it some current value.
XAnswer also has For and Let operations.

4.3.1 For operation

For operation defined as follows:

$$For^{var}_{A_i} A = \{z \mid z = Project_{A_i} A,\ x \in A\}$$

Here, *var* is a variable name; A_i – identifier of Envelop attribute; A - Envelop.

The result of evaluation of For operation is a new Envlelop, obtained from given by applying projection by attribute corresponding to A_i. The variable name became an identifier of the single attribute of the new Envelop.

Example 3

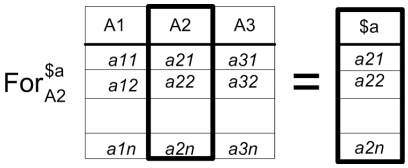

Figure 5: For operation

4.3.2 Let operation

Let operation defined as follows:

$$Let^{var}_{func_{i_1..i_n}} A = \{(x, z) \mid x \in A,\ z = func_{i_1..i_n}(x)\}$$

Here *var* is a variable name; func - a function referring to some already bound variables; $i_1..i_n$ – attribute identifiers corresponding to those variables. For example as this function could be an XPath expression like following: $b/address/country.

The result is a new Envelop, obtained by appending to the old one a new column with results of evaluation of function for each tuple. The given variable name became an identifier of appended attribute.

Example 4

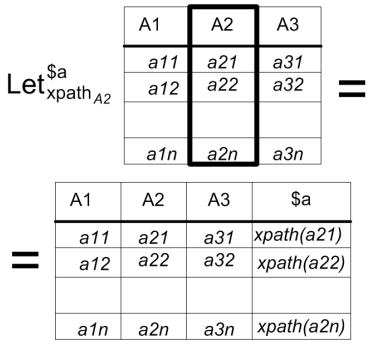

Figure 6: Let operation

In this case path expression depends on single variable corresponding to attribute denoted by $A2.

4.3.3 Return operation

The next one important algebraic operation, specific for XQuery, is a Return operation. XAnswer Return operation by functions and representation is similar to the XTasy Return operation [12, 5].
$Return_{OF} A$, where A - an Envelop produced with one of the XAnswer operations like Join or For operation. OF - Output Filter, which is defined by following rule:

OF ::= OF1,...,OFn | label[val] | @label[val] |val

val ::= var | varCopy | xpath.

Output Filter is a description of activities for representation of an evaluated data. For example it could be an XML element or an XML attribute constructor, or a variable value or a result of evaluation of navigational expression for some bound variable and so on.

4.3.4 DJoin operation

This operation is used when evaluation of one Envelop depends on evaluation of another. The only way to perform this operation is nested loops. Therefore the major goal of optimization is to translate it to another join operations whenever it is possible.

4.3.5 Examples

In the following example there are shown two algebraic expression for Xtasy and XAnswer algebras for the same XQuery expression.

Example 5

for $b in document("books.xml")/book
/author/addr
return <entry>$b</entry>

An Xtasy algebraic expression for this query:

$$Return_{entry[\$b]} path_{(\_,\$b,in)book[(\_,\_,/)author[(\_,\_,/)addr[\emptyset]]]}(books.xml)$$

An XAnswer algebraic expression:

$$Return_{entry[\$b]}((GetDocumentRoot(books.xml) \otimes_{child_{v_1 v_2}} GetElement(book, books.xml)) \otimes_{child_{v_2 v_3}} GetElement(author, books.xml)) \otimes_{child_{v_3 v_4}} GetElement(addr, books.xml)$$

Inspite of horizontal and vertical decomposition rules of path operation in Xtasy [12], the space of equivalent plans obtained in terms of XAnswer algebra is wider then in terms of XTasy.

Example 6

for $a in document(“doc.xml)/manufacturers
/dealer//address,
$b in document(“doc2.xml”)//manager
return $a

The tree of algebraic expression, corresponding to the given query is shown in Figure 7. For simplicity, nodes corresponding to join operation do not provide any information about predicates by which these join operations are performed. Copy of variable value operation is used as output filter for the return operation. As result a new element with different to current elements id is created. Also as output filter it could be a variable reference operation. In this case an element with the same id would be returned. This is the way to make changes in the document.

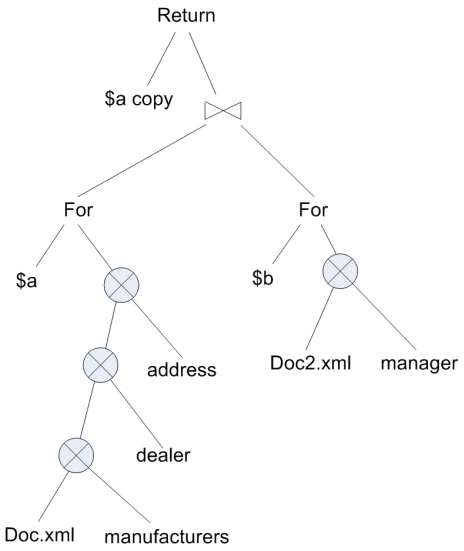

Figure 7: An algebraic expression tree for query from example 6

The next example shows a query with nested for-clauses where the inner has a dependency to the variable defined in the outer.

Example 7

for $a in document(“doc.xml”)//dealer,
$b in $a//address
return $b

A tree of algebraic expression for this case is shown in Figure 8. In this representation both for-clauses input filters have the same part, corresponding to the expression document(“doc.xml”)//dealer. In this case common sub-expression would be evaluated once and then obtained result would be reused in evaluation of the second for-clause.

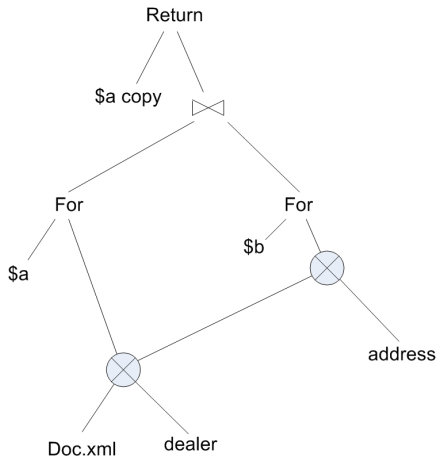

Figure 8: A tree of algebraic expression for example 7

4.4 Using some algebraic properties in query optimization

Sometimes nested for-clauses have not dependencies between themselves but they have similar parts of input path expressions. In this case special optimization technique could be applied. It obtains common parts of path expressions and rewrites expression to provide sharing result of common parts. Such query and corresponding to it algebraic tree after some reforming are described in Example 8. In [14] such optimization is called expression minimization. It is shown that benefit of this optimization for class of similar queries could reach 20-70%, depending on XML document structure. Each sub-query could have it's own optimal plans that have not common parts. In this case techniques derived from multi query optimization for building optimal plan for group of queries [11] could be used. In case of such queries the space of equivalent plans in terms of XAnswer algebra is wider than in terms of XAT.

Example 8

Lets consider following query: find title for those books author of that is a first author at least of one book.

Corresponding XQuery expression:

for $a in document("bib.xml")/book/author[1]
for $b in document("bib.xml")/book
where $b/author = $a
return $b/title

It is easy to see that expression document("bib.xml")/author could be evaluated once for first for-clause and then obtained result could be reused in evaluation of second for-clause. In this case algebraic expression could be presented as a graph, shown in Figure 9.

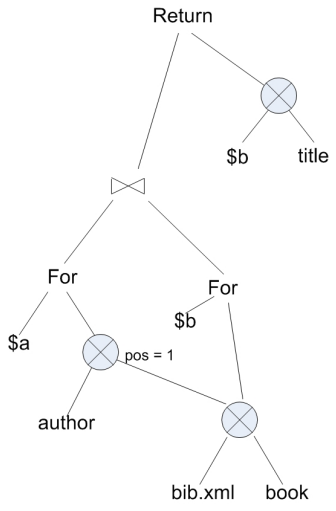

Figure 9: A graph of algebraic expression for example 8 after performing minimization

5 Conclusion

This paper outlines several requirements for query algebra suitable for building high-performance cost-based optimizer for native XML DBMS. Also it was shown that known algebras do not completely satisfy to these requirements. An another algebra that satisfies to these requirements better was introduced. This algebra is a base of XAnswer optimizer for native XML DBMSs which is currently under development.

References

[1] Monetdb home page http://monetdb.cwi.nl.

[2] S. Al-Khalifa, H. V. Jagadish, J. M. Patel, Y. Wu, N. Koudas, and D. Srivastava. "structural joins: A primitive for efficient xml query pattern matching". In 18th International Conference on Data Engineering (ICDE'02), 2002.

[3] M. Fernandez B. Choi and J. Simeon. The xquery formal semantics: A foundation for implementation and optimization. Technical report, World Wide Web Consortium, 2002.

[4] Catriel Beeri and Yariv Tzaban. SAL: An algebra for semistructured data and XML. In WebDB (Informal Proceedings), pages 37–42, 1999.

[5] C.Sartiani. Efficient Management of Semistructured XML Data. PhD thesis, Universita degli Studi di Pisa, 2003.

[6] Torsten Grust. Accelerating the xpath location steps. In ACM SIGMOD, pages 109–120, 2002.

[7] Torsten Grust, Maurice Van Keulen, and Jens Teubner. Accelerating xpath evaluation in any rdbms. ACM Trans. Database Syst., 29(1):91–131, 2004.

[8] H. V. Jagadish, Laks V. S. Lakshmanan, Divesh Srivastava, and Keith Thompson. Tax: A tree algebra for xml. In DBPL '01: Revised Papers from the 8th International Workshop on Database Programming Languages, pages 149–164, London, UK, 2002. Springer-Verlag.

[9] Ioana Manolescu, Daniela Florescu, and Donald Kossmann. Answering xml queries on heterogeneous data sources. In VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases, pages 241–250, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[10] Priti Mishra and Margaret H. Eich. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

[11] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, pages 249–260, New York, NY, USA, 2000. ACM Press.

[12] Carlo Sartiani and Antonio Albano. Yet another query algebra for xml data. In IDEAS '02: Proceedings of the 2002 International Symposium on Database Engineering & Applications, pages 106–115, Washington, DC, USA, 2002. IEEE Computer Society.

[13] S. Cluet V. Christophides and J. Simeon. Semistructured and structured integration reconciled: Yat += efficient query processing. Technical report, INRIA, Verso database group, 1998.

[14] Song Wang, Elke A. Rundensteiner, and Murali Mani. Optimization of nested xquery expressions with orderby clauses. In ICDEW '05: Proceedings of the 21st International Conference on Data Engineering Workshops, page 1277, Washington, DC, USA, 2005. IEEE Computer Society.

[15] J.Patel Y.Wu and H.Jagadish. Structural join order selection for xml query optimization. In ICDE ' 03 : International Conference on Data Engineering, 2003.