XQuery Optimization Based on Rewriting

Maxim Grinev

Moscow State University
Vorob'evy Gory, Moscow 119992, Russia
maxim@grinev.net

Abstract

This paper briefly describes major results of the author's dissertation work devoted to XQuery optimization based on rewriting.

1 Introduction

It is widely accepted doctrine that query languages should be declarative. As a consequence of this there may be several alternative ways to formulate a query. It is noticed that different formulations of a query can provide widely varying performance often differing by orders of magnitude. Relaying on the reasoning, sophisticated techniques for query transformations for relational query languages such as SQL were worked up [10, 11, 12, 7, 4, 5, 6]. The techniques allow rewriting a query into equivalent one that can be executed faster. The general characteristics of query rewriting techniques can be summarized as follows:

- The phase of rewriting optimization follows query parsing and precedes query plan generation.
- Rewriting optimization transforms a query into equivalent one.
- The selection of query transformations is carried out heuristically. Query transformations selected should be ameliorative for the majority of queries.
- Query rewriting is usually carried out on the basis of information obtained from the query itself, views to which the query is addressed, integrity constraints and the schema of data queried. The important note is that data and even statistic about data are not involved in query rewriting. They are used at the phase of cost-based plan optimization.

The emergence of XQuery [1] as a standard declarative language for querying XML data [3] calls for rewriting techniques that meet the same challenges as those for traditional query languages but that are developed in XQuery terms. The thesis is devoted to a comprehensive discussion of XQuery rewriting in the presence of data schema. The previous version of XQuery rewriting techniques described in this paper was published in [14].

2 Research Issues

Due to the properties of XQuery - such as powerful facilities for data transformation by means of support for XML element and XML attribute constructors and facilities to deal with incompleteness and irregularity of XML data - unmodified rewriting optimization techniques for traditional query languages are inappropriate. XQuery-specific and comprehensive rewriting optimization technique is an issue.

3 Kinds of optimizing query transformation

Extensive case study results in a number of query transformation kinds that improve the representation of the optimized query in the majority of cases and for which the cost is not required:

1. *Predicate push down XML element constructors* changes the order of operations to apply predicates before XML element constructors. It helps to reduce the size of intermediate results to which XML element constructors are applied. This kind of transformation is of great importance because XML element constructor is an expensive operation, the evaluation of which requires deep copy of XML tree constructed.
2. *Perform projection of transformation* is to compute in static accessors that are applied to XML element constructors. It allows avoiding redundant computation of costly XML element constructors.
3. *Predicate push down iterators* changes the order of operations to apply predicates before iterators. This kind of transformations is analogous to relational "predicate push down join".
4. *Transformation to more accurate formulation on the basis of the schema* is useful when a query is

rewritten by the user that has vague notion about XML document schema. Making query formulation more accurate allows one to avoid redundant data scanning that is peculiar to such queries.

5. *Make the query representation as declarative as possible* allows physical optimizer to widen search space with optimal execution plans.

6. *User-defined XQuery function inlining* makes the query more optimizable with respect to other kinds of transformation.

Developing XQuery rewriting optimizer we have used an approach of the general theory of rewriting. In context of the theory the optimizer are usually defined as a set of rewriting rules. All rewriting rules are defined in terms of some logical representation of a query. We took a logical representation similar to XQuery Core [2]. Rewriting rules form a rewriting system. A rewriting system can be investigated to determine its properties. The important property of a rewriting system is a normal form property. A normal form of a query in respect of a rewriting system is a query representation to which no rules of the rewriting system can be applied. A rewriting system has a normal form property if the application of rules to any query representation reduces it to a normal form. In the thesis the XQuery rewriting optimization problem is posed as follows. Build a rewriting system and prove its two properties: the normal form property and that the system accomplishes all kinds of transformations listed above.

To simplify the proving we decomposed the rewriting system into a number of subsystems that can be applied to a query consecutively. Each subsystem performs one or more kinds of transformation. Due to consecutive subsystems application the properties can be proved for each subsystem independently. The subsystems are as follows (the kinds of transformation accomplished by each subsystem are specified):

1. Function inlining (user-defined XQuery function inlining)

2. Type-based optimization (transformation to more accurate formulation on the basis of the schema)

3. Structural rewriting (predicate push down XML element constructors, perform projection of transformations)

4. Translation into the extended logical representation (make the query representation as declarative as possible)

The order in the list is the order of subsystems application during optimization. The following sections describe these subsystems.

4 Function inlining

The presence of calls to user-defined XQuery functions makes it difficult to optimize the query for the following reasons. First, the type-based optimization of a function body becomes less effective because its behavior depends a lot on actual parameters. As a consequence it may be impossible to infer the actual type of the function body and its subexpression. Second, the presence of several calls to one function prevents from pushing predicate down into the function body because changing function body for one of the calls changes the results of the other calls. Function inlining is a good idea to get over the difficulties. But function inlining of recursive function is a problem because it results in infinite loop. In general case it is impossible to break the loop without taking data into consideration. In the thesis we have proposed an algorithm for an important class of recursive functions that traverse XML tree going down at each recursive call. In the algorithm schema of XML data is used to keep track of the length of the path from the current nodes to the leaves of XML tree traversed.

5 Type-based optimization

The execution of many XQuery operations may lead to costly data scanning that are not necessary. The example of such operations is XPath step with the node test equals to "*". Support for such operations gives the user facilities to write compact queries to complex or irregular XML data. Type-based optimization is used to avoid redundant data scanning during processing queries with such operations. The main idea behind type-based optimization is to use static type inference on the basis of the XML data schema. The results of static type inference can be used to make the query more precise and avoid many of redundant data scanning. XQuery type-based optimization is now the most elaborated issue [8, 9]. In the thesis we have intoduced some additional type-based rewriting rules that had not been mentioned in the literature yet and a method to increase the accuracy of type inference of subexpression. It is proved that the rewriting system formed by type-based rules has the normal form property. The idea of the method is to insert if-clauses to branch the query and get subexpressions for which precise type can be inferred. Experiments showed that the application of the method essentially increases the efficiency of type-based optimization especially when functions with structural recursion are concerned.

6 Structural rewriting

Structural rewriting system implements query transformations of two kinds 1 and 2. In the thesis it is shown that degree of effectiveness of the transformations depends on the language subset. From this perspective XQuery can be divided in three disjoint

subsets: basic operations, identity-based operations, position-based operations (such as fn:contex-item(), fn:position(), fn:last() that are used to obtain information from the evaluation context). For queries that is composition of basic operations transformations of kinds 1 and 2 can be accomplished. For queries with identity-based operations and position-based operations it is proved that transformations of kinds 1 and 2 cannot be accomplished for all such queries because of language-inherent reasons.

The rewriting system for basic operations includes rules of four kinds:

1. Distribute computations (i.e. iterating over the sequence e1 , e2 is equivalent to the sequence of two iterations, one over e1 and one over e2)
2. Applying accessors to constructors (accessors that are applied to constructors can be rewritten into subexpressions of constructors).
3. Iterator applied to the singleton sequence (i.e. obtained as a result of constructor computation) can be replaced by the result of substitution of the singleton sequence for all occurrences of iterator variable in iterator body.
4. Static computation of effective boolean value, typed value, accessor value. This rules are based on the proposition that effective boolean value, typed value or accessor value of composition of meta-operations (such as iterator, if-operator, typeswitch, etc. that control evaluation of the expression) and XML element constructors can be rewritten into expression with the constructors replaced by other operations. It is carried out as follows. The expression, for which some value is to be computed, is traversed from operation to its parameters. When a meta-operation is encountered the traversal is continued. When XML element constructor is encountered, it is replaced with some other expression depending on what kind of value (effective boolean, typed or accessor) is computed. When some another operation different from meta-operations and XML element constructor is encountered, the traversal is stopped.

It is proved that the rewriting system has the normal form property and the following one.

Theorem 1 *In the normal form only the following operations in parameters marked "?" - $return(e, \lambda(x|?))$, $seq(?, \ldots, ?)$, $if(e, ?, ?)$, $ts(e, \lambda(x|cases(case(e, ?), \ldots, case(e, ?), def(e))))$, $element(e, ?)$ - can be applied to the results of XML element constructor*

Informally the theorem states the following. Only operations, that returns operands without any analysis of their content (maybe as a part of some new structure), can be applied to the XML objects that are XML elements constructed during the evaluation or that contain XML elements constructed during the evaluation as their part.

There are two immediate corollaries for the theorem:

1. All projections in the query are eliminated after rewriting. Assume that it is not so. It follows that there is an XML element constructed during the computation that does not present in the result. But it is a contrary to the statement of the theorem.
2. All predicates are pushed down XML element constructors. Assume that it is not so. It means that some predicates in the query is applied to an item that is an XML element or contains an XML element as its part. To check, whether the item satisfies the predicate, the content of the item must be analyzed. But it is a contrary to the statement of the theorem.

Extending the set of basic operations with position-based operations leads to the two problems. The first problem is that the results of the position-based operations depend on the context in which the operation is called. The context is implicitly generated. This breaks the referential transparency property of XQuery (that tends to be functional). The rewriting of query with position-based operation may result in non-equivalent query because the operation can change the position in the query and leave its context. To solve the problem a number of explicit *context generation operations* are introduced. The results of the context generation operations are bound to the iterator variables and all position-based operations are replaced with the variables. The rewriting rules for basic operations are extended to handle queries with context generation operations. Though using context generation operations allows one to rewrite many queries with position-based operations there are a number of query examples (with the last() operation) for which transformations of kinds 1 and 2 cannot be performed. This is the second problem that is language-inherent and cannot be solved by developing any rewriting system.

Extending the set of basic operations with identity-based operations leads to the locks preventing from perform transformations of kinds 1 and 2. In the thesis we partially solve the problem by introducing *UID generator*. UID is shorthand for Unique IDentifier. UID generator is an expression the result of which contains only nodes that have identifiers not equal to any another node involved in the computation. For example, any composition of XML element constructors is UID generator. Though using the notion of UID generator allows one to rewrite many queries with identity-based operations there are a number of query exam-

ples for which transformations of kinds 1 and 2 cannot be accomplished. This is a language-inherent problem that cannot be solved by developing any rewriting system.

7 Translation into the extended logical representation

Translation into the extended logical representation accomplishes the following two kinds of transformations: make the query representation as declarative as possible and predicate push down iterators. Making queries more declarative is well elaborated for relational query languages such as SQL [7, 4, 5, 6]. The major strategy used is to rewrite subqueries into joins because there are more options in generating execution plans for them. This increases possibility to find the most optimal execution plan. XQuery is not so declarative and there is less freedom in generating execution plans for the XQuery "join". It follows from the fact that XML items are ordered. Join in XQuery is expressed as nested iterators and the outer-most iterator determines the order of the result. It means that XQuery join doesn't commute as relational join does that doesn't allow evaluating join in various ways. But XQuery also supports for unordered sequences, which enables commutable joins. Rewriting subqueries (including those from predicate) can also simplify cost estimation procedure, query decomposition in data integration systems, multiple evaluation of self-contained subexpressions in nested iterators. In the thesis we extended the query representation with join operations such as ordered join, semijoin, and outerjoin and successfully adopted techniques for relational queries published in [6]. The techniques adopted allows rewriting any subqueries nested in predicates and nested iterators into the join operations. Also during the rewriting, the predicates are pushed down iterators. It is proved that the rewriting system has the normal form property. Besides we investigated techniques for determining the joins that can be evaluated without preserving the result order but it does not tell on the result of the whole query.

8 Conclusion and future work

Query optimization techniques described in this paper were fully implemented and integrated in virtual data integration system BizQuery [13]. Preliminary performance results have shown that these techniques improve query performance by orders of magnitude in a large number of common cases and the overheads incurred due to the query transformations are negligible compared with the time to execute complex queries.

Nevertheless the application of some rewriting rules (i.e. those used to perform predicate push down) might result in subexpressions propagation that in turn can lead to worse performance. Though experiments showed that it is not a common case, the main point in our future plan is to investigate such anomaly formally. Preliminary research of the anomaly leads to conclusion that applying techniques of finding common subexpressions can solve the problem.

References

[1] "XQuery 1.0: An XML Query Language." W3C Working Draft, 15 November 2002.

[2] "XQuery 1.0 and XPath 2.0 Formal Semantics." W3C Working Draft, 15 November 2002.

[3] "Extensible Markup Language (XML) 1.0 (Second Edition)." W3C Recommendation, 6 October 2000.

[4] W. Kim. "On Optimizing an SQL-like Nested Query", ACM Transactions on Database Systems, 7(3), September 1982.

[5] Richard A. Gansky and Harry K. T. Wong. Optimization of Nested SQL Queries Revisited. In Proc. ACM-SIGMOD International Conference on Management of Data, pages 23-33, 1987.

[6] Umeshwar Dayal. "Of Nests and Trees: A Unified Approach to Processing Queries that Contain Nested Subqueries, Aggregates, and Quantifiers", VLDB Conference, 1987.

[7] Hamid Pirahesh, Joseph M. Hellerstern, Waqar Hasan. "Extensible/Rule based Query Rewrite Optimization in Starburst", SIGMOD International Conference on Management of Data, 1992.

[8] P. Fankhauser. "XQuery Formal Semantics: State and Challenges.", SIGMOD Record 30(3): 14-19, 2001.

[9] B. Choi, M. Fernandez, J. Simeon. "The XQuery Formal Semantics: A Foundation for Implementation and Optimization.", www.cis.upenn.edu/ kkchoi/galax.pdf, 2002.

[10] M. Jarke, J. Koch. "Query Optimization in Database Systems." ACM Computing Surveys, Vol. 16, No. 2, June 1984.

[11] Y.E. Ioannidis. "Query Optimization." ACM Computing Surveys, Vol. 28, No. 1, 1996

[12] S. Chaudhuri. "An Overview of Query Optimization in Relational Systems." ACM PODS, 1998.

[13] Konstantin Antipin, Andrey Fomichev, Maxim Grinev, Sergey Kuznetsov, Leonid Novak, Peter Pleshachkov, Maria Rekouts, Denis Shiryaev. "Efficient Virtual Data Integration Based on XML." Submitted to ADBIS 2003.

[14] Maxim Grinev, Sergey D. Kuznetsov: Towards an Exhaustive Set of Rewriting Rules for XQuery Optimization: BizQuery Experience. ADBIS 2002: 340-345.