Let a Single FLWOR Bloom
(To Improve XQuery Plan Generation)*

Matthias Brantner, Carl-Christian Kanne, and Guido Moerkotte

University of Mannheim
{brantner,kanne,moerkotte}@informatik.uni-mannheim.de

Abstract. To globally optimize execution plans for XQuery expressions, a plan generator must generate and compare plan alternatives. In proven compiler architectures, the unit of plan generation is the query block. Fewer query blocks mean a larger search space for the plan generator and lead to a generally higher quality of the execution plans. The goal of this paper is to provide a toolkit for developers of XQuery evaluators to transform XQuery expressions into expressions with as few query blocks as possible.

Our toolkit takes the form of rewrite rules merging the inner and outer FLWOR expressions into single FLWORs. We focus on previously unpublished rewrite rules and on inner FLWORs occurring in the `for`, `let`, and `return` clauses in the outer FLWOR.

1 Introduction

XQuery evaluators become more and more mature in terms of features and performance, and XQuery is being integrated into mainstream DBMS products as a native language. However, XQuery processing research is still missing some fundamental tools to facilitate the development of industrial-strength XQuery optimizers. The goal of this paper is to fill one of these gaps: We provide a rewrite toolkit that allows to reduce the number of query blocks in a query expression. This widens the search space for plan generators by making more information visible to a single run of the plan generation algorithm. Let us begin by stressing the importance of our goal:

Industrial-strength query optimizers proceed in a two-phase manner. In a first phase, the query is translated into an internal representation, and heuristical rewrite rules are applied to simplify and normalize the query. In a second phase, a plan generator enumerates alternative execution plans, determines their costs, and chooses the optimal plan. Alternative plans can differ in the access paths used for the basic input sets (e.g. whether to use an index or not), in the order in which the basic input sets are joined, and in the position of other operators, such as grouping or sorting.

However, efficient plan generation algorithms cannot take arbitrary query structures as input. Instead, the unit of plan generation is the *query block*. Depending on the design of the query compiler, a query block can be represented in a variety of ways, for example as a source language construct (SELECT FROM WHERE in SQL, or FLWOR in XQuery), as a node in an internal graph representation (such as the Query Graph

* This work was supported by the Deutsche Forschungsgemeinschaft under grant MO 507/10-1.

D. Barbosa et al. (Eds.): XSym 2007, LNCS 4704, pp. 46–61, 2007.

Model QGM [17]), or as an algebraic expression. Some queries exhibit a nested structure, where a query block references subquery blocks. In such cases, the plan generator is called in a bottom-up fashion, generating plans for all subquery blocks before the surrounding query block is processed. It is easy to see that in such cases, the search space examined by the plan generator is limited, because only locally good solutions are computed. For globally optimal plans, it is desirable to reduce the number of query blocks to have more information available in a single run of the plan generator, creating a larger search space of alternative plans. For this reason, in the first phase of optimization, queries are rewritten by merging as many query blocks as possible. This is state-of-the-art for SQL query processing (e.g. [4,10,18]), but not highly developed for XQuery.

For an industrial-strength approach to XQuery optimization, such a rewriting step to merge query blocks is particularly necessary:

- In XQuery expressions in real applications, a nested query structure is the norm rather than an exception. This is due to a number of reasons, including the construction of hierarchical XML results, the absence of a grouping construct, the generation of queries using visual editors, and, last but not least, the inlining of (non-recursive) XQuery functions that contain FLWOR expressions.
- XML query processing can benefit from holistic n-way joins [3] which perform single-pass tree-pattern matching instead of constructing results just using binary joins. The detection of tree patterns and the decision when to use regular joins and when to use pattern matching is a global decision during plan generation that requires access to as much of the query as possible.

An example for a highly nested query (inspired by XMark Query 3) is shown here:

```
let $auction := doc("auction.xml") return
  let $euro:=for $o in $auction/site/open_auctions/open_auction
             for $i in $auction/site/regions/europe/item/@id
             where $o/itemref/@item eq $i
             return $o
  for $a in $euro
  where zero-or-one($a/bidder[1]/increase/text()) * 2
        <= $a/bidder[last()]/increase/text()
  return
    for $p in $auction/site/people/person[profile/@income > 5000]
    for $w in $p/watches/watch
    where $a/@id = $w/@open_auction
    return <auction id="{$a/@id}">
             <increase first="{$a/bidder[1]/increase/text()}"
                       last="{$a/bidder[last()]/increase/text()}"/>
               <watched_by id="{$p/@id}"/>
           </auction>
```

The query body is constructed of four FLWOR expressions, three of which are nested inside other FLWORs. However, these are only the explicit FLWOR blocks. Depending on the compiler design, the number of nested query blocks may be even deeper. For example, with a plan generator that focuses on purely structural tree-pattern matching, nested value-based predicates such as `profile/@income > 5000` may be separate query blocks.

Without further processing, such a query is optimized using several runs of the plan generation algorithm, where each plan for a FLWOR expression is used in the plan for