Efficient XQuery Evaluation of Grouping Conditions with Duplicate Removals

Norman May and Guido Moerkotte

University of Mannheim
B6, 29
68131 Mannheim, Germany
{norman,moer}@db.informatik.uni-mannheim.de

Abstract. Currently, grouping in XQuery must be expressed implicitly with nested FLWOR expressions. With XQuery 1.1, an explicit **group by** clause will be part of this query language. As users integrate this new construct into their applications, it becomes important to have efficient evaluation techniques available to process even complex grouping conditions. Among them, the removal of distinct values or distinct nodes in the partitions defined by the **group by** clause is not well-supported yet. The evaluation technique proposed in this paper is able to handle duplicate removal in the partitions efficiently. Experiments show the superiority of our solution compared to state-of-the-art query processing.

1 Motivation

XML gains importance as a storage format for business or scientific data. As more and more data is stored in this format, analytical query processing, i.e. XOLAP, becomes an important requirement. XQuery is the query language standardized for this purpose. While XQuery already includes a rich set of features, it lacks functions to support analytical query processing efficiently. Most importantly, grouping must be formulated implicitly with nested queries. While this challenge has already been addressed by techniques that unnest nested queries, end users and database implementors have identified the need for an explicit grouping construct that allows for efficient processing.

Consequently, a value-based grouping construct is part of the core requirements for XQuery 1.1, the next version of XQuery. Since the work on this version has just started, we will use the proposal for a **group by** construct by Beyer et. al. [1]. An efficient XQuery execution engine should include a powerful implementation of the grouping operator. With minor extensions of the relational grouping operators, it is possible to support several cases of grouping. We focus on a case that is neither well-supported for XQuery nor for SQL: We investigate efficient evaluation techniques for **group by** where duplicates are removed on different attributes of tuples that are in the same partition.

D. Barbosa et al. (Eds.): XSym 2007, LNCS 4704, pp. 62–76, 2007.

1.1 Motivating Example

Consider the following example query on the XMark document instance depicted in Fig. 1(a). It counts for every open auction the total number of bidders, the number of distinct bidders, the maximum increase, and the number of different increases (We abbreviate some element or attribute names as follows: personref – pr, @person – @p, increase – i).

```
for $auction in $doc/site/open_auctions/open_auction,
    $bidder  in $auction/bidder
let $person   := $bidder/personref/@person,
    $increase := $bidder/increase
group by $auction into $a using fn:deep−equal
      nest $person   into $p,
           $increase into $i
      let $pd := distinct−values($p),
          $pi := distinct−values($i)
return
  <status>
       { $a/seller }
       <bid−count>       { count($p)  } </bid−count>
       <distinct−bidders> { count($pd) } </distinct−bidders>
       <max−increase>    { max($i)    } </max−increase>
       <distinct−steps>  { count($id) } </distinct−steps>
  </status>
```

In the example query, the keyword **group by** is directly followed by the *grouping expression* (a reference to variable $auction), the keyword **into**, and the *grouping variable* ($a). The function mentioned after the keyword **using** is used to partition the input tuples into groups. The *nesting expression* after the keyword **nest** is applied to every tuple that is assigned to some group. The result of this computation is appended to the current group referenced by the *nesting variable*. In the example query, we use the optional **let** clause to remove duplicates from the sequences bound to the nesting variables. The result of the **group by** is a sequence of tuples that contains bindings for every grouping variable and every nesting variable. Notice that sequence order is not meaningful in the context of this operator because there is no immediate relationship between input tuples and output tuples any more. We denote with *aggregation variable* all variables in the scope of the **return** clause that are the argument of an aggregate function, i.e. $p, $i, $pd, $pi in our example query.

Fig. 1(b) shows the tuples that serve as input to the **group by**. We trace how the group defined by a single open auction, a_1, is processed. In Fig. 1(c), we have identified groups – in this example, there is only a single group. To compute the result of the aggregate functions in the **return** clause of this query, we need to remove duplicate values from every sequence bound to the nesting variables. We have highlighted them in the two sequences.

Most systems try to avoid copying and, hence, would filter duplicates by discarding complete input tuples. In general, this only works when duplicates are removed from at most one aggregation variable. In our example, however,