Supporting Efficient Grouping and Summary Information for Semistructured Digital Libraries*

Minsoo Lee[1], Sookyung Song[1], Yunmi Kim[1], and Hyoseop Shin[2,**]

[1] Department of Computer Science and Engineering
Ewha Womans University, Seoul, Korea
mlee@ewha.ac.kr, happymint@ewhain.net, cherish11@ewhain.net
[2] Department of Internet and Multimedia Engineering
Konkuk University, Seoul, Korea
hsshin@konkuk.ac.kr

Abstract. XML is the most popular platform-independent data expression language which is used to specify various digital content such as web content, multimedia content, bio-chemical data, etc. These various forms of XML data are continuously increasing by a large amount and there is a strong demand on effectively managing such data in digital libraries or archives. The most popular query language to search and retrieve information from such semi-structured XML digital libraries is XQuery. XQuery has a very powerful syntax which allows users to iterate over data items and perform calculation, string matching, and output formatting. However, it lacks a simple and easy way to group and provide summaries on vast amounts of XML data. This grouping and summary function is especially important for large digital archives where users like to obtain an overview or summary of the contents in the digital library. Our work is focused on providing an easy way for grouping in XQuery at the query language level. We provide several cases where this can be considered to be effective. We have also implemented an XQuery processing system with grouping functions based on the eXist Native XML Database.

1 Introduction

XML(eXtensible Markup Language) is increasingly becoming popular as a data expression and exchange language on the Web as well as a specification language for various multimedia content due to its flexibility and platform-independence[1]. Large amounts of digital content represented in XML format are becoming increasingly available and there is strong demand to provide digital libraries and archives for such XML data. The W3C has already adopted XML as a standard and there have been several languages devised to query XML data. Among such query languages XQuery is now considered the standard[2]. The XQuery language uses a FLWR(For-Let-Where-Return) syntax to devise queries. XQuery uses XPath[3] expressions to specify

* This work was supported by the Korea Research Foundation Grant (KRF-2004-041-D00572) and also partially supported by the second stage of the BK21 program.
** Corresponding author.

S. Sugimoto et al. (Eds.): ICADL 2006, LNCS 4312, pp. 400 – 409, 2006.

hierarchical relationships, and has powerful iteration capabilities as well as convenient calculation and formatting functions.

However, the current XQuery provides very poor functionality regarding grouping and summary (i.e., aggregation) capabilities on XML data. This is especially important in large digital libraries, where users would frequently ask for an overview or summary on the large amount of XML data stored in digital libraries. Because XQuery requires users to specify their grouping and aggregation queries as multiple nested structures and join operations, it is hard to express and understand such complex query statements. In this paper, we propose a way to extend XQuery at the language level to enable such grouping and aggregation queries to be much more easily formulated. Considering various cases for grouping in semistructured data, we extend the EBNF of XQuery to incorporate a group by clause. In addition, we extended the eXist native database system[4] to implement our idea and validate the usefulness of our approach.

The organization of the paper is as follows. Section 2 discusses related research on grouping support for queries on XML data. Section 3 gives an overview of our XQuery extension to support the groupby clause as well as several cases where this could greatly benefit query construction in digital libraries. Section 4 deals with the implementation of our system based on the eXist native database and gives an explanation on how the grouping and aggregation is processed in our system. Section 5 gives the conclusion and discusses future work.

2 Related Research

Grouping on data requires restructuring of the original data and enables related data to be treated together as a group, thus allowing aggregations to be computed on the groups. Research on grouping data is still an issue in relational databases. Especially in data warehouses complex analytical queries containing various grouping conditions are issued. Therefore, users need a way to easily specify the grouping methods. Several new operators for grouping have been suggested in the data warehouse research area [5,6].

Grouping in XML documents is a much more serious issue yet it is considered as a more difficult problem due to the semistructured nature of the XML documents. There has been some work that helped understanding the difference between relational and XML data in terms of grouping and provided insight on proposing grouping operators for XML[7]. In this work, they have focused on providing binding variables for a set of tuples instead of individual tuples and devised a GApply operator that can be integrated into existing relational database engines. The Lore semistructured database system[8] does not support the groupby clause and requires a complex query to be formulated to perform grouping tasks. The TIMBER project[9] defined a tree algebra called TAX to internally identify the grouping information and used it to transform a nested XQuery into a TAX grouped query. Deutsch et al.[10] extend tree pattern queries into Group-by Normal Form Tree Pattern(GNFTP) queries, which are nested, perform arbitrary joins, and freely mix bag and set

semantics. They describe a subset of XQuery, called OptXQuery and provide a normalization algorithm that rewrites any OptXQuery into a GNFTP query. The algorithm detects and eliminates redundant navigation within and across nested subqueries and it unifies and generalizes prior solutions for tree pattern minimization and group-by detection. Beyer et al.[11] provide a proposal for extending the XQuery FLWOR expression with explicit syntax for grouping and for numbering of results. In this work, they show that these new XQuery constructs not only simplify the construction and evaluation of queries requiring grouping and ranking but also enable complex analytic queries such as moving-window aggregation and rollups along dynamic hierarchies to be expressed without additional language extensions.

3 Extension of XQuery with Groupby Clause

This section explains the XQuery extension to incorporate the groupby clause. We show how our approach is consistent with the syntactic and semantic specification of XQuery by first giving the EBNF(Extended Backus-Naur Form) that includes the groupby clause and then discussing the specific query types that could benefit from the use of the groupby clause. Figure 1 shows only the extended EBNF part of the XQuery language specification including the groupby clause.

```
FLWGRExpr  ::=  (ForClause|LetClause)+ WhereClause?
                    groupbyClause?   "return"  ExprSingle
ForClause      ::= <"for" "$"> VarName TypeDeclaration? PositionalVar? "in" ExprSingle
                    ("," "$" VarName TypeDeclaration? PositionalVar? "in" ExprSingle)*
LetClause      ::= <"let" "$"> VarName TypeDeclaration? ":=" ExprSingle ("," "$" VarName
                    TypeDeclaration?":=" ExprSingle)*
WhereClause  ::= "where" ExprSingle
groupbyClause  ::= <"groupby"> groupbySpecList
groupbySpecList  ::=  ( NgroupbySpec | SgroupbySpec )
SgroupbySpec ::= "[" groupbySpec ("," groupbySpec)* "]"
NgroupbySpec ::= groupbySpec ("," groupbySpec)*
groupbySpec  ::= ExprSingle
```

Fig. 1. EBNF including groupby in XQuery

The following subsections show 4 types of queries that could benefit from the use of the groupby clause in XQuery and are based on the XML example data shown in Figure 2.

3.1 XQuery Type 1: Group by with Single Binding Variable

XQuery Type 1 is a basic example for using group by with a single binding variable. Using the XML in Figure 2, the following query which groups according to the "author" information could be easily formulated with a groupby clause extension.

- Type 1: Output the book *titles* grouped by the *author* who published the book.

```
<?xml version="1.0" encoding="utf-8"?>
<bib>
   <book year="1994">
      <title>TCP/IP Illustrated</title>
      <author>Stevens W.</author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
   <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
      <author>Stevens W.</author>
      <author>Abiteboul Serge</author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
   <book year="2000">
      <title>Data on the Web</title>
      <author>Abiteboul Serge</author>
      <author>Buneman Peter</author>
      <author>Suciu Dan</author>
      <publisher>Morgan Kaufmann Publishers</publisher>
      <price>39.95</price>
   </book>
   <book year="1999">
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>CITI</editor>
      <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>
</bib>
```

Fig. 2. Example XML data

XQuery Type 1: Without groupby

```
<results>
 {
   let $a := doc("bib.xml")//author
 ⓐ for $author in distinct-values($a/text())
   return
  <result>
    <author>{ $author }</author>
    <titles>  {
     ⓑ for $b in doc("bib.xml")/bib/book
          where some $ba in $b/author
         satisfies ($ba/text() = $author)
          return $b/title
              }
    </titles>
  </result>
 }
</results>
```

XQuery Type 1: With groupby

```
<results>
 {
for $b in doc("bib.xml")/bib/book,
    $a in $b/author
group by  $a
return
  <result>
    { $a }
    <titles>
      { $b/title }
    </titles>
  </result>
 }
</results>
```

Fig. 3. Comparison of XQuery Type 1 with and without groupby clause

When using the current XQuery syntax to devise such a query it becomes very complex and requires a nested XQuery to obtain the desired result. However, by including a group by clause it becomes very easy to formulate such a query. Figure 3 shows this difference in formulating the query.

3.2 XQuery Type 2: Group by Used with Aggregation Function

XQuery Type 2 applies aggregation functions on the grouped data. Using the XML in Figure 2, the following query calculates the "total number of books" in the groups.

- Type 2: Output the book *titles* and *total number of books* grouped by the *author* who published the book.

Again, when using the current XQuery syntax, a nested XQuery is needed to obtain the desired result. A group by clause can simplify the query. Figure 4 shows the difference when using the current XQuery without the groupby clause and when using the extended XQuery with the groupby clause to express such a query.

XQuery Type 2: Without groupby	XQuery Type 2: With groupby

<results>
{
let $a := **doc**("bib.xml")//author
ⓐ **for** $author in distinct-values($a/text())
let $t :=
ⓑ **for** $b **in doc**("bib.xml")/bib/book
where some $ba **in** $b/author
satisfies ($ba/text()= $author)
return $b/title
return
<result>
<author>{ $author }</author>
<title-count> { count($t) }</title-count>
<titles>{ $t }</titles>
</result>
}
</results>
	<results>
	{
	for $b **in doc**("bib.xml")/bib/book,
	$a **in** $b/author
	group by $a
	return
	<result>
	{ $a }
	<title-count> { count($b/title) }
	</title-count>
	<titles> { $b/title }
	</titles>
	</result>
	}
	</results>

Fig. 4. Comparison of XQuery Type 2 with and without groupby clause

3.3 XQuery Type 3: Group by with Two or More Binding Variables

XQuery Type 3 groups the XML data based on two or more binding variables. The following query is an example using the XML data shown in Figure 2.

- Type 3: Output the book *titles* grouped by the *author* who published the book and the *year* in which the book was published.

The current XQuery syntax requires a deeply nested XQuery to obtain the desired result, whereas a group by clause enables it to be expressed in a single level query. Figure 5 shows this difference.

XQuery Type 3: Without groupby

```
<results>
 {
  let $a := doc("bib.xml")//author
ⓐ for $author in distinct-values($a/text())
  return
  <result>
    <author>{ $author }</author>
    {
    let $year :=
     doc("bib.xml")/bib/book[author=
     $author]/year
  ⓑ for $y in distinct-values($year/text())
     return
      <year-titles>
        <year>{ $y }</year>
        <titles>
        {
       ⓒ for $b in doc("bib.xml")/bib
          /book[author=$author and
          year=$y]
        return
          $b/title
        }
        </titles>
      </year-titles>
    }
  </result>
 }
</results>
```

XQuery Type 3: With groupby

```
<results>
 {
for $b in doc("bib.xml")/bib/book,
    $a in $b/author,
    $y in $b/@year
group by $a, $y
return
   <result>
      { $a }
     <year-titles>
       <year>  { $y } </year>
       <titles> { $b/title }  </titles>
     </year-titles>
   </result>
 }
</results>
```

Fig. 5. Comparison of XQuery Type 3 with and without groupby clause

3.4 XQuery Type 4: Group by with Binding Variable Composed of Set of Values

XQuery Type 4 groups the XML data based on a binding variable that represents a set of values. Using the example XML data shown in Figure 2, the following query could be easily formulated with a groupby clause extension.

- Type 4: Output the book *titles* grouped by the *set of authors* who published the book.

This kind of query will take into consideration "a set of elements" instead of individual values and use it to group other information. Figure 6 shows the difference when using the current XQuery without the groupby clause and when using the extended XQuery with the groupby clause to express such a query.

XQuery Type 4: Without groupby

```
<results>
{
① let $au1 := ⓐ for $b1 in doc("bib.xml")/bib/book
          where exists($b1/author)
          return  <author-set> {
                    for $a1 in $b1/author
                    order by $a1
                    return $a1  }
                  </author-set>
② let $au2 := ⓑ for $b2 in doc("bib.xml")/bib/book
                  where exists($b2/author)
                  return   <author-set> {
                             for $a2 in $b2/author
                             order by $a2
                             return $a2   }
                           </author-set>
③ let $au3 := union($au1, $au2)
ⓒ for $au4 in $au3/author-set
  return
    <result>
      { $au4 }
      <titles>  {
       ⓓ for $b in doc("bib.xml")/bib/book
         where exists($b/author)
         let $au := ⓔ let $au5 := $b/author
                      return  <author-set> {
                                for $a3 in $b/author
                                order by $a3
                                return $a3  }
                                  </author-set>
         where deep-equal($au,$au4)
         return  $b/title }
      </titles>
    </result>
}
</results>
```

XQuery Type 4: With groupby

```
<results>
{
for $b in doc("bib.xml")/bib/book
let $a := $b/author
group by [ $a ]
return
     <result>
       <author-set> { $a }
       </author-set>
    <titles>
      { $b/title }
    </titles>
    </result>
}
</results>
```

Fig. 6. Comparison of XQuery Type 4 with and without groupby clause

4 Implementation of XQuery Processor Supporting Groupby Clause

We have implemented a prototype query processor that supports XQuery with the group by clause extension to demonstrate the feasibility of our approach. Performance issues will be pursued in future work. The prototype is implemented using the eXist native database system[4]. The development environment is as follows. For the server, eXist 1.0 was used, and jEdit 4.2 was used as the client. Eclipse SDK 3.0.2 was used as the Integrated Development Environment(IDE). JDK 1.4.2 and XQuery 1.0 were used. The overall architecture of the system is shown in Figure 7. The

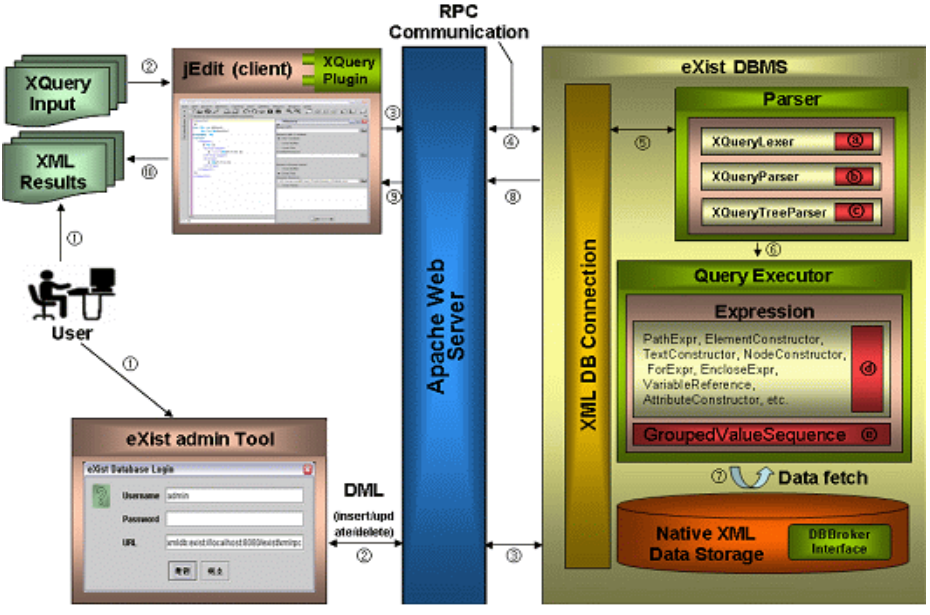


Fig. 7. Overview of architecture of the XQuery with group by processor

XQueryLexer, XQueryParser, XQueryTreeParser, and various parts of the Query Executor of the eXist system, shown as (a)-(e) in Figure 7, were modified.

Figure 8 gives a high-level overview on processing the grouping and aggregation queries. The query is first parsed and then executed and an initial XML DOM tree result without the grouping is obtained. The grouping elements are then searched and group keys are assigned to each group element. Afterwards, the XML DOM tree is restructured according to the grouping information and output format. During this step the aggregation values are also computed. The final XML result is then returned to the user.

A simple example of the three major steps are shown in Figure 9. The first step is identifying the grouping elements and assigning the group key elements. The initial

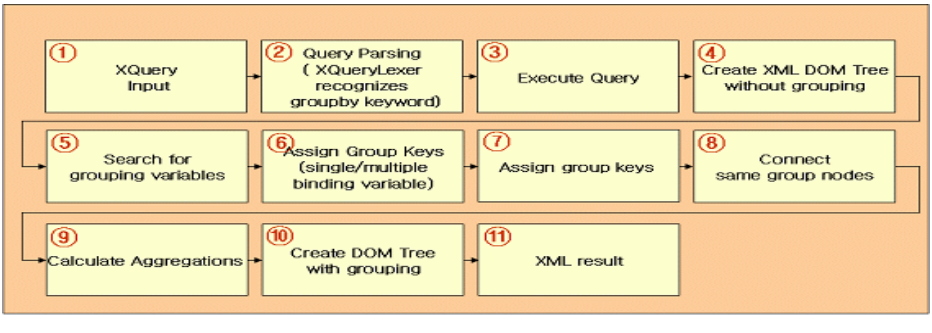
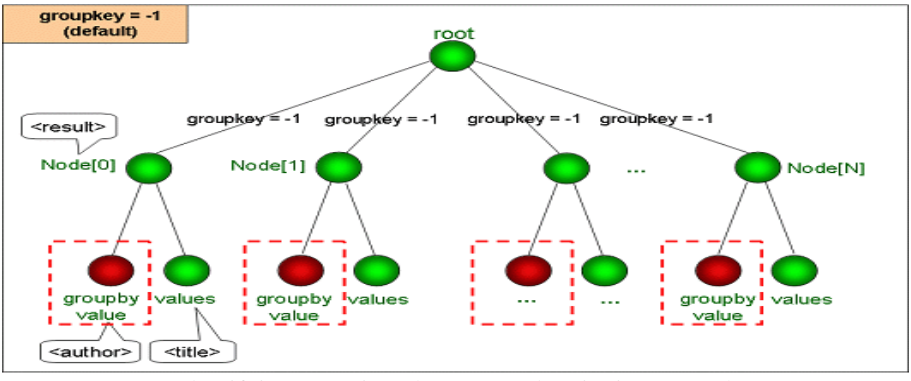
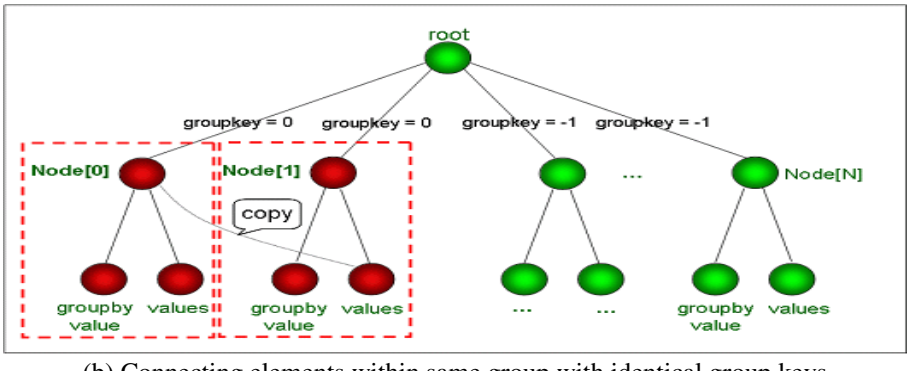


Fig. 8. Overview of processing XQuery with group by feature

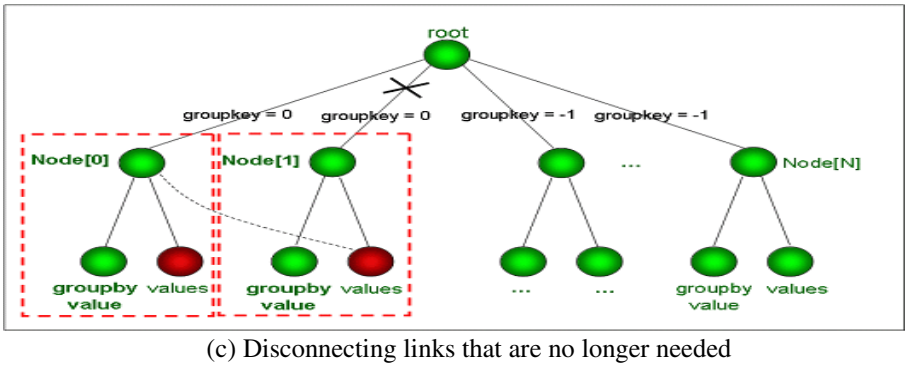
value of the group key is –1, but when groups composed of more than one participating elements are identified, the group key value is assigned with a positive integer. The second step is to connect the elements within the same group to restructure the XML DOM tree. The third step disconnects those prior links that are no longer needed after elements are grouped together.



(a) Identifying grouping elements and assigning group keys



(b) Connecting elements within same group with identical group keys



(c) Disconnecting links that are no longer needed

Fig. 9. Three major steps in processing groups in XQuery

5 Conclusion

In this paper, we have proposed an extension to the XQuery language to effectively support grouping and summaries on XML data. This work enables users of semi-structured digital libraries to easily formulate queries that provide overview or summary information for a large size digital library. We have also implemented the processing of XQuery with the groupby clause on the eXist native database system. The contribution of our paper is that with this extension to XQuery, the complex and nested equivalent queries would be reduced to simple XQueries using this groupby clause. Some comparisons on other work are shown in Table 1. Future work include performing extensive query optimization with group information and identifying more flexible semantics in group concepts regarding the semistructured nature of XML.

Table 1. Comparison of XQuery groupby support with other systems

	Lore	Timber	GApply	XQuery with groupby
Native XML support	△	○	×	○
XQuery support	×	△	×	○
Explicit group by support	×	△	○	○
Single group by variable support	×	○	○	○
Group by with aggregation support	×	×	×	○
Multiple group by variable support	×	○	○	○

○ full support △ partial support × no support

References

[1] XML(eXtensible Markup Language), http://www.w3.org/XML/

[2] XQuery (XML Query Language), http://www.w3.org/XML/Query/

[3] XML Path Language (XPath) 2.0, http://www.w3.org/TR/2005/WD-xpath20-20050404/

[4] eXist(An Open Source Native XML Database), http://exist.sourceforge.net/

[5] D. Chatziantoniou and K. A. Ross, “Querying multiple features of groups in relational databases," VLDB, 1996

[6] D. Chatziantoniou and K. A. Ross, “Groupwise processing of relational queries," VLDB, 1997

[7] S. Chaudhuri, R. Kaushik and J.F. Naughton, “On Relational Support for XML Publishing: Beyond Sorting and Tagging”, SIGMOD, 2003

[8] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom, “Lore: A Database Management System for Semistructured Data”, SIGMOD Record, 26(3):54-66, September 1997

[9] H. V. Jagadish, Shurug Al-Khalifa, Adriane Chapman, Laks V.S. Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, Nuwee Wiwatwattana, Yuqing Wu and Cong Yu. “TIMBER: A Native XML Database”, VLDB Journal, Vol. 11, Issue 4, 2002

[10] Alin Deutsch , Yannis Papakonstantinou , Yu Xu, “Minimization and Group-By Detection for Nested XQueries”, Int’l Conference on Data Engineering (ICDE) , pp. 839, 2004

[11] Kevin Beyer, Don Chamberlin, Latha S. Colby, Fatma Ozcan, Hamid Pirahesh, Yu Xu, “XML query, update, and search: Extending XQuery for analytics", Proceedings of the 2005 ACM SIGMOD Int’l Conference on Management of Data, pp. 503-514, June 2005