XQuery Rewriter

CSE 636 Fall 2008 Project Phase 2

November 14, 2008

In Phase 2, you will incorporate a query rewriter to your query processor in order to evaluate XQuery expressions efficiently. You will implement an algorithm which detects the fact that the FOR and WHERE clause computation can be implemented using a join operator. You may assume that the input XQuery expressions to be optimized are in the simplified XQuery syntax also given below. No need to first normalize your queries to this form. Your query rewriter should just print out the rewritten XQuery expression. No need to implement the join operator and execute the rewritten query.

You may team up with at most one partner for this phase.

1 Applying Database Optimization Techniques on XQuery

The XQuery processor built in the first phase of the project misses the query optimization opportunities we know from the database literature. This may lead to unacceptable performance, as the following example shows.

EXAMPLE 1 Consider the following query.

```
for $b in doc("input")/book,
    $a in doc("input")/entry,
    $tb in $b/title,
    $ta in $a/title
where $tb eq $ta
return
    <book-with-prices>
    {    $tb,
         <price-review>{ $a/price/text() }</price-review>,
         <price>{ $b/price/text() }</price>
    }
    </book-with-prices>
```

Assume the query is evaluated on an input file with 10^5 books and 10^5 reviews. Then in the brute force evaluation of the plan the where clause will be evaluated 10^{10} times, which is unacceptable from a performance point of view, since there is a much more efficient way to evaluate the query. In particular, one may:

1. collect a tuple set B, consisting of all tuples (b, t_b) of books and their titles
2. collect a tuple set E, consisting of all tuples (a, t_a) of entries and their titles
3. join the tuple sets B and E on the titles and derive a new tuple set R consisting of tuples (b, t_b, a, t_a)
4. produce a `book-with-prices` element for every tuple of R.

The above plan can employ efficient join methods for Step 3. For example, one may index the tuples of B by title and then for each tuple of E find matching tuples of B using the index. Other efficient join methods can also be used (e.g., sort merge join).

Introducing A Join Operator Let us introduce a new XQuery operator, called join that

1. inputs two lists of tuples. Each tuple is of the form

   ```
   <tuple>
      <a1>v1</a1>
      .
      .
      .
      <an>vn</an>
   </tuple>
   ```

   where the strings `a1,...,an` are the attribute names. Each attribute value `v1,...,vn` is a list of XML elements in the general case. The tuples of each list are homogeneous, in the sense that all tuples have the same attributes.

2. inputs two lists of attribute names, originating in the two tuple lists. We augment the XQuery syntax with a "list of constants" notation. In particular, we denote the list of attributes `a1,...,an` as `[a1,...,an]`.

3. and outputs a list of tuples, where the output order is non-specific.

Using the join operator we can write more efficient versions of our queries, as the following example shows.

EXAMPLE 2 We rewrite the query of Example 1, using the join operator.

```
for $tuple in join(for $b in doc("input")/book,
                       $tb in $b/title
                   return <tuple> <b> {$b} </b> <tb> {$tb} </tb> </tuple>,

                   for $a in doc("input")/entry,
                       $ta in $a/title
                   return <tuple> <a> {$a} </a> <ta> {$ta} </ta> </tuple>,

                   [tb], [ta]
                  )
return
    <book-with-prices>
    {    $tuple/tb,
         <price-review>{ $tuple/a/price/text() }</price-review>,
         <price>{ $tuple/b/price/text() }</price>
    }
    </book-with-prices>
```

The join operator above has four arguments. The first two arguments deliver the book tuples and entry tuples, while the third and fourth arguments specify that the join is on attributes `tb` and `ta`.

Yet Another Subset of XQuery Consider the following subset of XQuery, where there are no nested FWR expressions and hence it is easier to introduce join operations.

$$\begin{array}{rcl}
XQuery & \rightarrow & \textsf{for } V_1 \textsf{ in } Path_1,\ ...,\ V_n \textsf{ in } Path_n \\
 & & \textsf{where } Cond \textsf{ return } Return \\
 & & \\
Path & \rightarrow & (\textsf{doc}(FileName)|Var)/n_1/.../n_m \\
 & | & (\textsf{doc}(FileName)|Var)/n_1/.../\textsf{text}() \\
 & & \\
Return & \rightarrow & Var \\
 & | & Return_1, Return_2 \\
 & | & < n > \{Return\} < /n > \\
 & | & Path \\
 & & \\
Cond & \rightarrow & Var_1 \textsf{ eq } (Var_2|Constant) \\
 & | & Cond_1 \textsf{ AND } Cond_2
\end{array}$$

Queries of the above subset can be rewritten to make efficient use of the join operator, as the following example shows.

EXAMPLE 3 The following query returns triplets of books, where the first book has a first author named John, the second book has a common author with the first book and a common author with the third book. Notice that the query below may generate multiple triplets with the same books - even triplets where the same three books appear in the same order.

```
for $b1 in doc("input")/book,
    $aj in $b1/author/first/text(),
    $a1 in $b1/author,
    $af1 in $a1/first/text(),
    $al1 in $a1/last/text(),
    $b2 in doc("input")/book,
    $a21 in $b2/author,
    $af21 in $a21/first/text(),
    $al21 in $a21/last/text(),
    $a22 in $b2/author,
    $af22 in $a22/first/text(),
    $al22 in $a22/last/text(),
    $b3 in doc("input")/book,
    $a3 in $b3/author,
    $af3 in $a3/first/text(),
    $al3 in $a3/last/text()
where $aj eq "John" AND
      $af1 eq $af21 AND $al1 eq $al21 AND
      $af22 eq $af3 AND $al22 eq $al3
return <triplet> {$b1, $b2, $b3} </triplet>
```

Notice how the rewriting uses two joins. Notice also that the joins are on pairs of attributes. The first join is on the pairs of attributes `[af1, al1]` and `[af21, al21]`. The second join is on the pairs of attributes `[af22, al22]` and `[af3, al3]`.

```
for $tuple in join(
                join(for $b1 in doc("input")/book,
                         $aj in $b1/author/first/text(),
                         $a1 in $b1/author,
                         $af1 in $a1/first/text(),
                         $al1 in $a1/last/text()
```

```
                          where $aj eq "John"
                          return <tuple> <b1>{$b1}</b1>
                                         <af1>{$af1}</af1>
                                         <al1>{$al1}</al1>
                                 </tuple>,

                          for $b2 in doc("input")/book,
                              $a21 in $b2/author,
                              $af21 in $a21/first/text(),
                              $al21 in $a21/last/text(),
                              $a22 in $b2/author,
                              $af22 in $a22/first/text(),
                              $al22 in $a22/last/text()
                          return <tuple> <b2>{$b2}</b2>
                                         <af21>{$af21}</af21>
                                         <al21>{$al21}</al21>
                                         <af22>{$af22}</af22>
                                         <al22>{$al22}</al22>
                                 </tuple>,

                           [af1, al1], [af21, al21]
                          ),

                    for $b3 in doc("input")/book,
                        $a3 in $b3/author,
                        $af3 in $a3/first/text(),
                        $al3 in $a3/last/text()
                    return <tuple> <b3>{$b3}</b3>
                                   <af3>{$af3}</af3>
                                   <al3>{$al3}</al3>
                           </tuple>,

                    [af22, al22], [af3, al3]
                 )
return <triplet> {$tuple/b1, $tuple/b2, $tuple/b3} </triplet>
```

Implementation Ideas There are multiple algorithms that may lead to rewritings such as the ones described above. The key component of all algorithms is the analysis of navigations performed in the `in` clauses and of the conditions in the `where` clause. A useful tool would be to construct a labeled graph that represents the input XQuery expressions and captures the navigation and the conditions. For example, the graph in Figure 1 illustrates the navigation and condition structure of the query of Example 3. The nodes are labeled with variables or the input document, while the solid edges are labeled with the navigations that connect the variables of the nodes. The dotted edges correspond to the conditions of the where clause.

The task of locating the joins can be viewed as a task of locating partitions, as shown in Figure 2 that connect via the dotted edges, which, in effect, become the join conditions.

2 Deliverables

Due Date	Command Line	Output Format
Sunday, December 7th	XQueryRewriter xqueryfile.xq	Your XQueryRewriter class should print out the rewritten XQuery.

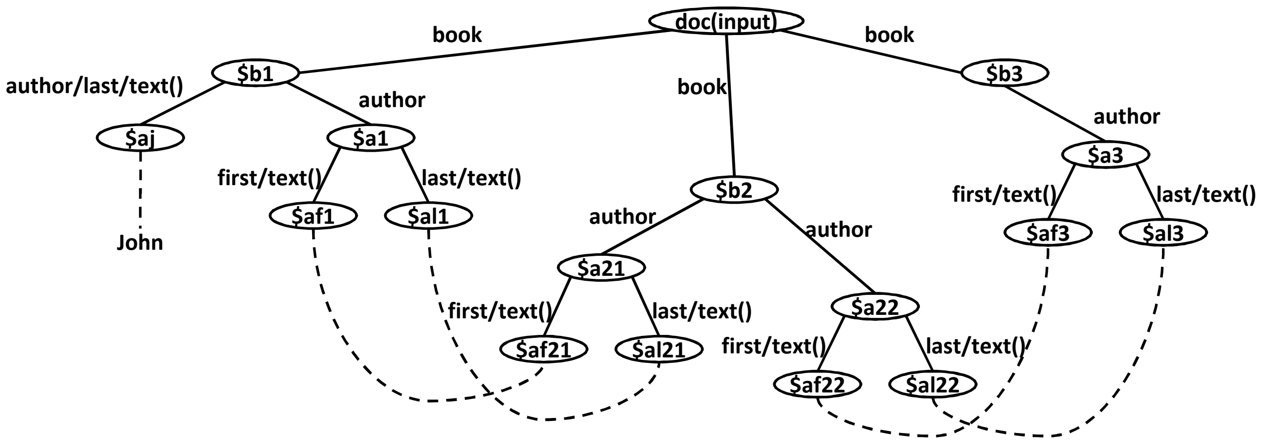

Figure 1

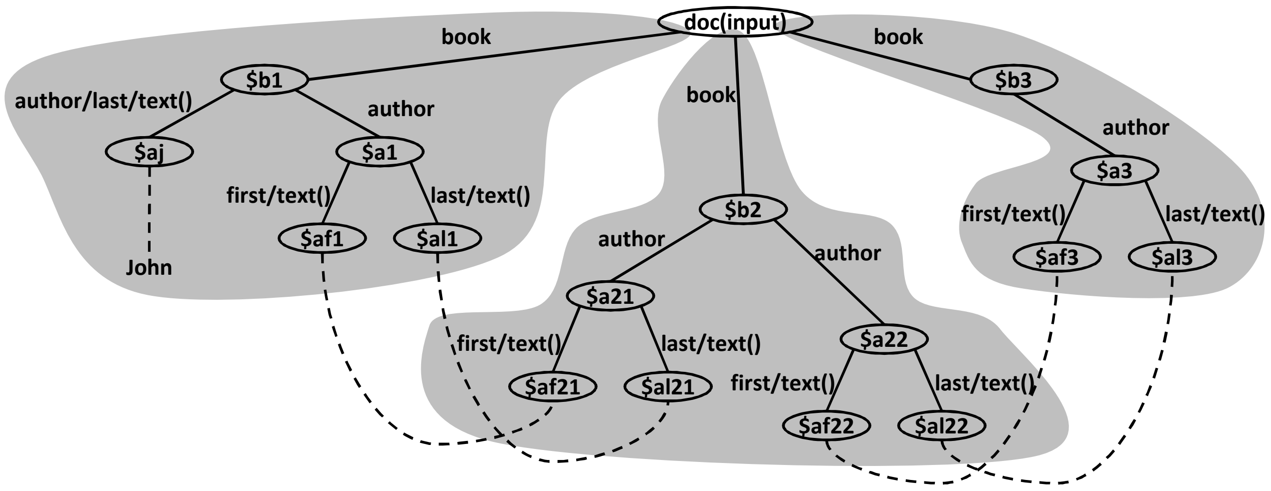

Figure 2

Only one of the team members should email the instructor a .zip archive containing the following items:

- A README.txt file describing the archive contents.
- A PDF file describing your implementation.
- The source code you wrote in folder **src** and the compiled code in folder **bin**.

Important The Java class mentioned in the "Command Line" column above should be located within the **bin** folder.