

## Implementation and Optimization

The purpose of this exercise is to familiarize you with query plans and optimization of query plans for XQuery.

- For a better understanding, it is useful to use the **mxquery** engine. You can visualize the query plan by giving the following parameters:  

```
java -jar mxquery.jar queryfile.xq -printExpTree
```

 The expression tree is written in an XML form, which you can copy and paste into an XML Editor (Oxygen), to display the structure of the query tree.
- In this exercise we will display query trees using a similar notation with that of the mxquery engine, because the mxquery engine already does some optimizations and hides some details or compacts the query plan. The notation we use is explained in the “Introduction” section below. The differences between this notation and the notation of mxquery will also be explained in section “Introduction: Query Plans”. You can use any notation you feel appropriate.
- The requirements for this Exercise are written in Exercise 1 (“Implementation”) and 2 (“Optimization”).

### Introduction: Query Plans

The solution of this exercise is based on the following conventions:

<p><b>Let/return:</b> (a <i>return</i> belongs to a <i>let</i> or <i>for</i> )</p> <pre>let \$result := &lt;order&gt;ABC&lt;/order&gt; return \$result</pre> <p><b>Notes:</b> we represent the let variable as a label on the left edge; the right edge gives the result;          - the right edge (corresponding to the initial <b>return</b> ) cannot be executed before the left edge.</p>	
<p><b>For :</b></p> <pre>for \$a in doc('flights.xml')//Airport let \$c:= 5 return \$result &lt;result&gt;{\$c}&lt;/result&gt;</pre> <p><b>Notes:</b>          - The for variable is a label of the left edge. The left edge evaluates to a sequence of nodes.          - For each mapping of \$a to one of the items in the sequence generated by the left size, the right edge is executed          - the result of the for is the union of all results</p>	

<p>returned on the right side for each mapping  - the return belongs to the let statement, so the result returned at each iteration is the right edge of the <i>let</i> node.  A branch is read bottom-up: the “flights.xml” is used, then the doc is applied to it, then we evaluate descendant-or-self on this node, for getting the “Airport”, and the result is bound to \$a</p>	
<p><b>Evaluating conditions: ifthenelse</b></p> <p><b>Example:</b></p> <p><b>let</b> \$results := [...] -- not displayed for space reasons  <b>return</b> \$results/result[count = 2]</p> <p><b>Note:</b> this Path expression is equivalent to a FLWR expression:</p> <pre>for \$r in \$results/result where count = 2 return \$r</pre> <ul style="list-style-type: none"> <li>- a temporary variable, \$r, is introduced</li> <li>- the condition is expressed using an <b>ifthenelse</b> node. An <b>ifthenelse</b> takes a Boolean expression, and returns the result of the second branch when the expression is true, and the result of the third branch when the result is false (in our case, we return the empty sequence)</li> </ul>	

**Notation Equivalence between the previous Notation and => the mxquery engine:**

0. \$variable\_name => VariableIterator (“variable\_name”)
1. \$results – children – match(“result”) => ChildrenIterator (VariableIterator(“results”), “result”)  
This means ChildrenIterator is a root node, with two subnodes: one is the VariableIterator of \$results, the other is the constant “result”.
2. \$doc – descendant-or-self – “Flight” => DescendantOrSelfIterator(VariableIterator(“\$doc”, “Flight”))
3. Inside for: ifthenelse => WhereIterator
4. any function call is represented as a FunctionCallIterator.
5. **NodeConstructor** => XMLContent

## Exercise 1: Implementation

Draw the query plan for the following queries:

### Query 1: [from Exercise Sheet 3]

Retrieve the list of the busiest airports on the date of '2005-12-24' (based on the number of departures and arrivals).

Note:

```
let $results := (<order>
{
for $a in doc('flights.xml')//Airport
let $c := count(doc('flights.xml')//Flight[((source eq $a/@airId) or
(destination eq $a/@airId)) and (date eq '2005-12-24') ] )
return
  <result>
    {$a}
    <count>{$c}</count>
  </result>
}
</order>)
return $results/result[count eq max($results//count)]
```

### Query 2: [from Exercise Sheet 3]

Retrieve all flight possibilities from ,North Pole' to ,SouthPole', on the date of 2006-12-24 with one or two intermediate stops.

- **Note: This query is not supported by the mxquery engine. Write the query plan using the notation above.**

```
for $f1 in doc('flights.xml')//Flight[source eq 'NPL'],
    $f3 in doc('flights.xml')//Flight[destination eq 'SPL'],
    $f2 in doc('flights.xml')//Flight
where $f1/destination = $f2/source
    and $f2/destination = $f3/source
    and xs:time($f1/arrival) lt
        xs:time($f2/departure)
    and xs:time($f2/arrival) lt
        xs:time($f3/departure)
    and $f1/seats > 0 and $f2/seats > 0 and $f3/seats > 0
return <possib>
    {$f1}
    {$f2}
    {$f3}
</possib>
```

**Query 3: [from Exercise Sheet 4]**

*Give for each author, the number of books he has written.*

```
for $a in distinct-values(doc("bib.xml")//author)
return <res>
  <name>{$a}</name>
  <count>
    {count(doc("bib.xml")//book[author= $a]) }
  </count>
</res>
```

**Exercise 2: Optimization**

- a) Estimate the execution cost of the previous queries (find a suitable measure for expressing the cost, e.g., number of nodes in the query plan which are used during execution).
- b) Optimize the query plans of the queries above, and show, using the chosen metric, that the rewritten plan is better.