XML Technologies

XQuery: Evaluation

Christian Grün
Database & Information Systems Group
Universität Konstanz
XQuery Evaluation

Introduction

- our expression tree will now be *evaluated*
- all nodes are recursively traversed and replaced by their computed value
XQuery Evaluation

Challenge

- conventional evaluation may produce huge intermediate results
- final result is possibly much smaller

Example: \((1 \text{ to } 1000000)[1]\)
XQuery Evaluation

Iterative processing

- idea: request, process and pass on query results one by one
- main advantage: constant memory usage
- alternative wording: streaming, pipelining

Lazy evaluation

- computation of an expression is delayed until its value is required
- some values in an expression may not need to be computed at all

💡 How many items need be computed for \((1 \ to \ 1000000)[1]\)?
XQuery Evaluation

Iterator pattern

- instead of `Evaluate()`, expressions get a `Next()` function, which returns a single result item
- a `null` value is returned if an expression is *exhausted*, i.e., if all values have been returned

Alternative

- expressions can be equipped with an `Iterator()` function, which creates and returns a new iterator
- advantage: the expression itself will be *immutable* (need not be reset, etc.)
XQuery Evaluation

Iterator examples

**IndexOf.Iterator.Next() : Item**

---

**Require:**
- ITERATOR := iterator on the input sequence ($\text{seq}$)
- ITEM := item to be found ($\text{item}$)
- POS := 0 (current iterator position)

```xml
1     loop
2       POS := POS + 1
3       item := ITERATOR.Next()
4     if item = null then
5       return null
6     else if ITEM equals item then
7       return new Integer(POS)
8   end if
9 end loop
```

**fn:index-of function**

---

**Intersect.Iterator.Next() : Node**

**Require:**
- ITERATORS := array with iterators on all operands

```xml
1     nodes := array with cached nodes
2   for i := 0 to #ITERATORS-1 do
3     nodes[i] := ITERATORS[i].Next()
4     return null if nodes[i] is null
5   end for
6     i := 1
7   while i < #nodes do
8     diff := nodes[0].ID - nodes[i].ID
9     if diff < 0 then
10    nodes[0] := ITERATORS[0].Next()
11   return null if nodes[0] is null
12    i := 1
13 else if diff > 0 then
14    nodes[i] := ITERATORS[i].Next()
15   return null if nodes[i] is null
16   else
17     i := i + 1
18   end if
19 end while
20 return nodes[0]
```

**intersect operator (for ordered input nodes)**
# XQuery Evaluation

**Iterator: meta data**

The result size of an expression can be help to speed up evaluation:

- **Size()** returns the number of iterated items, or **-1** if the number is unknown
- **Get(i)** retrieves the specified item
- **Get-Remaining()** returns remaining (possibly all) items
- **Reset()** resets the iterator

## Example

```plaintext
Count.Item() : Item

Require: EXPR := argument of the count function
1    iter := EXPR.Iterator()
2    c := iter.Size()
3    if c = -1 then
4        repeat
5            c := c + 1
6        until iter.Next() = null
7    end if
8    return new Integer(c)
```

**fn:count** function, handling both known and unknown sizes
XQuery Evaluation

Blocking operators

- not all expressions can be evaluated in an iterative way
- *blocking operators* require all items to be cached before the actual result can be computed and returned

Examples

- for $i$ in 1 to 100 order by $i$ descending return $i$
- for $i$ in 1 to 5 let $j := "A"$ group by $j$ return $j$
- /descendant::A/descendant::B

Can you explain why these expressions are *blocking*?
XQuery Evaluation

Location path blocking

Optimized node caching:
- cache all incoming nodes
- only sort nodes, and remove duplicates, if necessary:
  - \( \text{diff} = 0 \): new node equals last entry \( \rightarrow \) node is ignored
  - \( \text{diff} < 0 \): new node appears before last entry \( \rightarrow \) ORDERED flag is invalidated

```
NodeIterator.Add(Node n)

Require:
  NODES := array with cached nodes
  ORDERED indicates if nodes are ordered and duplicate-free
1  iter := EXPR.Iterator()
2    c := iter.Size()
3  if ORDERED and #NODES > 0 then
4     diff := n.ID - NODES[#NODES - 1].ID
5     if diff = 0 then
6        return
7     else if diff < 0 then
8           ORDERED = false
9      end if
10    end if
11  add n to NODES
```
XQuery Evaluation

Iterative processing: summary

- iterative query evaluation is an elegant approach to process large data sets
- it facilitates both \( constant \) memory usage and \( faster \) execution times
- iterative processing is limited by \( blocking \) operators
- iteration slows down querying if values are requested \( multiple \) times

\[\text{best results are gained by combining iterative and caching algorithms}\]
\[\text{all existing information on data (number of results, positional access) should be utilized!}\]