Relational Algebra: Mother Tongue XQuery: Fluent

How to Compile XQuery Expressions into a Relational Algebra

Torsten Grust Jens Teubner

University of Konstanz
Dept. of Computer and Information Science
Konstanz, Germany

June 21, 2004

Relational XPath Back-Ends

Relational databases can efficiently back XPath evaluation.

- Encode XML structure using a numbering scheme.
  - "XPath accelerator" with *pre*/*post* tuples.
- Re-use existing database technology.
  - Storage management, index structures, query optimization.
- Support XPath evaluation through tailor-made operators.
  - "Staircase join" exploits properties of the *pre*/*post* encoding.
  - "Holistic" join algorithms to process entire XPath expressions.

➠ **Compile XQuery expressions into Relational Algebra.**

XQuery is More than XPath

Important XQuery features are not yet covered.

- Iteration in XQuery `FLOWR` expressions.
  - Contrary to set-oriented processing in relational databases?
- Construction of transient XML tree nodes.
  - Document table must be "extended" during query processing.

This talk addresses the first issue.
(The paper additionally covers the second.)
The ideas are orthogonal to efficient XPath evaluation.

XQuery FLWOR Expressions

XQuery is built around a looping primitive, the `for` construct.

$$\texttt{for \$}v\ \texttt{in}\ (x_1,\ x_2, \ldots,\ x_n)\ \texttt{return}\ e$$
$$\equiv$$
$$(\ e[x_1/\$v],\ e[x_2/\$v], \ldots,\ e[x_n/\$v]\ )$$

- $\$v$ is successively bound to the values of x_i.
- The return expression e is evaluated for each binding.
- XQuery is a functional-style language.
  ➠ It is sound to evaluate e for all bindings in parallel.

The XQuery Data Model

XQuery's basic data type is the sequence.

- Any expression evaluates to an **ordered sequence of items**.
- Sequences are always **flat**.

We may represent a sequence using a two-column table.

pos	*item*
1	`"a"`
2	`"b"`
3	`"c"`

- Encode order in *pos*, the atom value in *item*.

(For now, we assume a polymorphic `item` type.)

Loop Lifting for Constant Subexpressions

We extend our sequence encoding by the column *iter* that accounts for the independent iterations.

iter	*pos*	*item*
1	1	10
1	2	20
2	1	10
2	2	20
3	1	10
3	2	20

Example:

`for` $\$v_0$ `in (1,2,3) return 10`

→ 10 appears in 3 iterations.

`for` $\$v_0$ `in (1,2,3) return (10, 20)`

→ `(10, 20)` appears in 3 iterations.

We refer to this as the **loop lifted** representation of a sequence.

Deriving a Loop Lifted Value Representation

We derive a compilation procedure that solely operates on loop lifted sequences.

Example: Body of `for $v in (10, 20, 30) return $v`

iter	*pos*	*item*
1	1	10
2	1	20
3	1	30

- Start with representation of `(10, 20, 30)`.
- Generate a new iteration for each value.
- Each value forms a singleton sequence. ($pos = 1$ for each tuple)

The **row number** operator ϱ in our algebra generates unique *iters*.

Nested XQuery Expressions

XQuery allows arbitrary expression nesting.

- **Example:**

$$s \left\{ \begin{array}{l} \texttt{for } \$v_0 \texttt{ in (1,2) return} \\ s_0 \left\{ \begin{array}{l} \texttt{for } \$v_{0\cdot 0} \texttt{ in (10,20) return} \\ s_{0\cdot 0} \{ \; (\$v_0, \$v_{0\cdot 0}) \end{array} \right. \end{array} \right.$$

- We need to **map** value representations between scopes.
  - Variables defined in surrounding scopes.
  - The expression result of the `for` expression.

Mapping Between Scopes

We capture expression nesting with help of a relation map.

- **Example:** (previous slide)

s {
```
for $v0 in (1,2) return
    for $v0.0 in (10,20) return
        ($v0,$v0.0)
```
s_0 { for $\$v_{0\cdot0}$ …, $s_{0\cdot0}$ { ($\$v_0$, $\$v_{0\cdot0}$)

outer	*inner*
1	1
1	2
2	3
2	4

$\text{map}_{(0,0\cdot0)}$

- The relation lists corresponding *iter* values of a `for` body and its surrounding scope.
- Value mapping then renders into a join with this relation.
- We handle **arbitrary** nesting this way.

Translatable Subset of XQuery Core

We support an XQuery subset that suffices to handle the XMark benchmark set.

literals	`42`, `"foo"`, `()`, `...`
arithmetics	e_1 `+` e_2, e_1 `-` e_2, `...`
builtin functions	`fn:sum(`e`)`, `fn:count(`e`)`, `fn:doc(`*uri*`)`
variable bindings	`let $`v `:=` e_1 `return` e_2
iteration	`for $`v `at $`p `in` e_1 `return` e_2
conditionals	`if` p `then` e_1 `else` e_2
sequence construction	e_1 `,` e_2
function calls	f `(`e_1`,` e_2`,` `...,` e_n`)`
element construction	`element` e_1 `{` e_2 `}`
XPath steps	$e/\alpha::\nu$

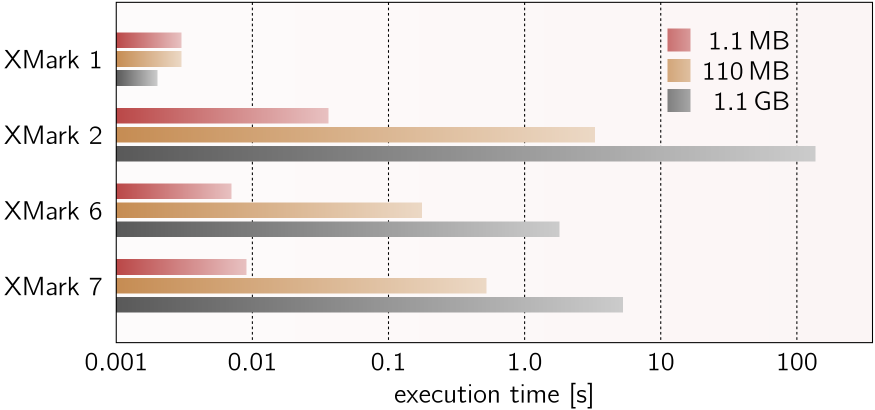
A Relational Algebra to Evaluate XQuery

We generate a (almost) standard Relational Algebra.

$\pi_{a_1:b_1,\ldots,a_n:b_n}$	projection/renaming
σ_a	selection
$\dot\cup$	disjoint union
$\times$	Cartesian product
$-$	difference
$\bowtie_{a=b}$	equi-join
$\varrho_{b:\langle a_1,\ldots,a_n\rangle/p}$	row numbering
$⏌_{\alpha,\nu}$	XPath axis join
ε	element construction
$\circledast_{b:\langle a_1,\ldots,a_n\rangle}$	n-ary arithmetic/comparison operator $*$
$\underline{a\,\vert\,b}$	literal table
$count, \ldots$	count and other aggregation functions

XMark on DB2

We implemented our translation for XMark queries in SQL.



Summary

We propose a fully relational evaluation for XQuery.

- A compilation procedure translates XQuery expressions into Relational Algebra.
- We can handle `FLWOR` expression, element construction and others.
- We can deal with arbitrary expression nesting.
- Experiments with an SQL based DBMS are promising.
- This work is part of our ongoing project "Pathfinder."
- Resulting algebra expressions, however, can get large. . .