**XQuery**

- Query language for XML data
- includes XPath functionality (path steps)
- iterators, bindings, functions, ...
  (for, if then else, let, sequence, count, doc, ...)
- typing (cast, typeswitch, instance-of, ...)
- xml data construction (element, attribute, ...)

**Example:**

*How many sold items cost more than 40?*

```xml
count (for $i in doc("auction.xml")/site/
    closed_auctions/closed_auction
  where $i/price/text() >= 40
return $i/price)
```
Evaluating XQuery expressions with MonetDB

Jan Rittinger
CWI Amsterdam & University of Konstanz

Summerschool GK University of Konstanz
St. Vigil (Italy)
Motivation

- about me
  - studying in Konstanz
  - Bachelor thesis in database and information systems department (pathfinder project)
  - currently working at the Centrum voor Wiskunde en Informatica (CWI) in Amsterdam (the Netherlands)
  - database and multimedia retrieval department

- project: MonetDB
- plan for pathfinder: using a database for the query execution
- my task at the CWI: building a first XQuery prototype based on pathfinder using MonetDB
MonetDB  ([http://monetdb.cwi.nl/](http://monetdb.cwi.nl/))

- Main-Memory database
- main focus on cache conscious algorithms
- uses only binary tables
- open source software
- low level interpreter language (MIL)
  - good extensibility
    (simple interface for c-functions)
  - database operators
  - type system
  - iterative operators (e.g batloop, while, ...)
  - user-defined functions
idea

• take pathfinder and start from the typed XQuery Core representation

⇒ core tree

... relational operators

• solution: XQuery for SQL-Hosts
  (Grust, Sakr, Teubner - VLDB 2004)

• represent every intermediate result by a table with the schema iter|pos|item
the iter|pos|item schema

```
<table>
<thead>
<tr>
<th></th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>
```

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

- iter is needed to support the looping primitive `for`
  ➔ that is why everything with this schema is called loop-lifted
understanding the example

let $a := (10,20)$ return
for $b$ in $(30,40)$ return $(a,b)$

- the sequence $(10,20)$ is evaluated ...
- ... and bound to variable $a$

- the sequence $(30,40)$ is evaluated ...
- ... and each value of the sequence is iteratively bound to variable $b$

- for each binding the return sequence with the actual variable bindings is evaluated
understanding the example

```plaintext
let $a := (10,20)\ return
for $b$ in $(30,40)$ return ($a,$b)
```

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>40</td>
</tr>
</tbody>
</table>
translation of constants

let $a := (10,20)$ return
for $b$ in (30,40) return ($a$,$b$)

- for each constant a new table is created
- the table is expanded to the current scope (cross-product with loop$_0$)

<table>
<thead>
<tr>
<th>iter</th>
<th>loop$_0$</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>loop$_0$</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>
translation of sequences

let $a := (10,20)$ return
for $b$ in $(30,40)$ return ($a,$b)

- a (special) union is made to combine the two inputs
- the pos column gets new dense numbers within each iter

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>
translation of let expressions

let $a := (10,20)$ return
for $b$ in (30,40) return ($a,b$)

- the variable is added to the environment $\Gamma$ with the value of its expression

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>20</td>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>

\(a := \)

\[ \begin{array}{c|c|c|c|}
\hline
\text{iter} & \text{pos} & \text{item} \\
\hline
1 & 1 & 10 \\
1 & 2 & 20 \\
\hline
\end{array} \rightarrow \begin{array}{c|c|c|c|}
\hline
\text{iter} & \text{pos} & \text{item} \\
\hline
1 & 1 & 10 \\
1 & 2 & 20 \\
\hline
\end{array} \]
let $a := (10,20)$ return
for $b$ in $(30,40)$ return ($a,$b)
Let $a := (10,20)$ return
for $b$ in (30,40) return ($a$, $b$)

- $b$ is successively bound to the values (30, 40) - The return expression ($a$, $b$) is evaluated for each binding
- XQuery is a functional-style language
  ➔ It is sound to evaluate ($a$, $b$) for all bindings in parallel

\[
\begin{array}{ccc}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 30 \\
2 & 1 & 40 \\
\end{array}
\] ➔

\[
\begin{array}{ccc}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 30 \\
2 & 1 & 40 \\
\end{array}
\]

\[\text{let } b := \]

\[
\begin{array}{ccc}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 30 \\
2 & 1 & 40 \\
\end{array}
\] $b$
translation of for expression (2/3)

let $a := (10,20)$ return
for $b$ in $(30,40)$ return $(a,b)$

• every variable in the environment ($a$) needs to be mapped to the new scope

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>40</td>
</tr>
</tbody>
</table>

$\Rightarrow$

<table>
<thead>
<tr>
<th>iter</th>
<th>loop₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>outer</th>
<th>inner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Let $a := (10,20)$ return
for $b$ in $(30,40)$ return ($a$, $b$)

• every variable in the environment ($a$) needs to be mapped to the new scope

\[
\begin{array}{c|c|c}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 10 \\
1 & 2 & 20 \\
\end{array}
\quad \quad \quad
\begin{array}{c|c|c}
\text{outer} & \text{inner} \\
1 & 1 \\
1 & 2 \\
\end{array}
\quad \quad \quad
\begin{array}{c|c|c}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 10 \\
1 & 2 & 20 \\
2 & 1 & 10 \\
2 & 2 & 20 \\
\end{array}
\]

$|x|$ \quad \quad map_1 \quad $a$
translation of the for body

let $a := (10,20)$ return
for $b$ in (30,40) return ($a,$b)
backmapping of the result

let $a := (10,20)$ return
for $b$ in $(30,40)$ return ($a,b$)

$|x| = \begin{array} \text{inner} & \text{outer} \\ 1 & 1 \\ 2 & 1 \end{array}$

$\map_1$

$\begin{array}{r|rr}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 10 \\ 1 & 2 & 20 \\ 1 & 3 & 30 \\ 2 & 1 & 10 \\ 2 & 2 & 20 \\ 2 & 3 & 40 \\
\hline
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 10 \\ 1 & 2 & 20 \\ 1 & 3 & 30 \\ 1 & 4 & 10 \\ 1 & 5 & 20 \\ 1 & 6 & 40 \\
\end{array}$
conclusion

• almost every feature of XQuery is supported
• 18 of 20 Xmark queries are running with generated MIL code
• developed some new ideas, which work nicely
• discovered some problems/bugs in the pathfinder code

but ...

• performance problems (mostly because of iterative translations)
  ➔ solution: intelligent loop-lifted version
• a lot of work has still to be done (e.g. built-in functions, ...