MonetDB/XQuery (2/2):
High-Performance, Purely Relational XQuery Processing

http://pathfinder-xquery.org/
http://monetdb-xquery.org/

Stefan Manegold
Stefan.Manegold@cwi.nl
http://www.cwi.nl/~manegold/
XPath evaluation (SQL)

Example query:
/descendant::open_auction[./bidder]/annotation

SELECT DISTINCT a.pre
FROM doc r, doc oa, doc b, doc a
WHERE r.pre=0
  AND oa.pre > r.pre AND oa.post < r.post  <- descendant
  AND oa.name = "open_auction" AND oa.kind = "elem"
  AND b.pre > oa.pre AND b.post < oa.post
  AND b.level = oa.level + 1
  AND b.name = "bidder" AND b.kind < "elem"
  AND a.pre > oa.pre AND a.post < oa.post
  AND a.level = oa.level + 1
  AND a.name = "annotation" AND a.kind = "elem"
ORDER BY a.pre

• (potentially?) expensive joins due to range predicates
• (potentially?) expensive duplicate elimination
Staircase Join \[\text{VLDB03}\]

\text{pre} | \text{post} \text{ are not random numbers:}
\implies \text{exploit the tree properties encoded in them}
$\text{pre} \mid \text{post}$ are not random numbers:

=> exploit the tree properties encoded in them
Staircase Join \[\text{VLDB03}\]

\[\text{pre} \mid \text{post} \text{ are not random numbers:} \]
\[\Rightarrow \text{ exploit the tree properties encoded in them}\]
pre|post are not random numbers:
=> exploit the tree properties encoded in them
Staircase Join  [VLDB03]

- **skipping**: avoid touching node ranges that cannot contain results

Generate a duplicate-free result in document order
- **pruning**: reduce the context set a-priori
- **partitioning**: single sequential pass over the document

Diagram:
- seek → scan → skip → seek → scan → skip → ...
- List of context nodes
- Document
Staircase Join: Skipping

Example:

\[(c_1, c_2) / \text{descendant:}*\]

SELECT DISTINCT doc.pre
FROM c, doc
WHERE doc.pre > c.pre
AND doc.post < c.post

Avoid comparing large chunks of the document table.
Example:

\[(c_1, c_2, c_3, c_4)/\text{ancestor:}*\]

```
SELECT DISTINCT doc.pre
FROM c, doc
WHERE doc.pre < c.pre
  AND doc.post > c.post
```

Eliminate: \(c_1, c_3\)

Keep: \(c_2, c_4\)
Staircase Join: Partitioning

Example:

\[(c_1, c_2, c_3)/\text{ancestor}:*\]

SELECT DISTINCT doc.pre
FROM c, doc
WHERE doc.pre < c.pre
    AND doc.post > c.post

Single-pass algorithm that avoids generating duplicates
Example:

\[(c_1, c_2, c_3)/\text{ancestor:}*\]

```
SELECT DISTINCT doc.pre
FROM c, doc
WHERE doc.pre < c.pre
AND doc.post > c.post
```

Single-pass algorithm that avoids generating duplicates.
Schedule

• So far:
  • RDBMS back-end support for XML/XQuery (1/2):
    • Document Representation (*XPath Accelerator, Pre/Post plane*)
    • XPath navigation (*Staircase Join*)
Schedule

• So far
  • RDBMS back-end support for XML/XQuery (1/2):
    • Document Representation (*XPath Accelerator, Pre/Post plane*)
    • XPath navigation (*Staircase Join*)

• Now:
  • XQuery to Relational Algebra Compiler:
    • Item- & Sequence- Representation
    • Efficient FLWoR Evaluation (*Loop-Lifting*)
    • Optimization
  • RDBMS back-end support for XML/XQuery (2/2):
    • Updateable Document Representation
Source Language: XQuery Core

XQuery is a lot more than just XPath.

<table>
<thead>
<tr>
<th>Source Language: XQuery Core</th>
<th>42, &quot;foo&quot;, (), ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>literals</td>
<td>arithmetics</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>builtin functions</td>
<td>fn:sum(e), fn:count(e), fn:doc(uri)</td>
</tr>
<tr>
<td>variable bindings</td>
<td>let $v := e₁ return e₂</td>
</tr>
<tr>
<td>iteration</td>
<td>for $v at $p in e₁ return e₂</td>
</tr>
<tr>
<td>conditionals</td>
<td>if p then e₁ else e₂</td>
</tr>
<tr>
<td>sequence construction</td>
<td>e₁, e₂</td>
</tr>
<tr>
<td>function calls</td>
<td>f (e₁, e₂, ..., eₙ)</td>
</tr>
<tr>
<td>element construction</td>
<td>element e₁ { e₂ }</td>
</tr>
<tr>
<td>XPath steps</td>
<td>e/α::ν</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
Target Language: Relational Algebra

- **RDBMS kernels implement \( q \) in terms of SQL’s DENSE_RANK.**

- Most conceivable implementations of \( q \) require a sorted input.
• *sequence* = *table of items*
• *add pos column for maintaining order*
Sequence Representation

- Item sequences, sequence order

(10, “x”, <a/>, 10) →

<table>
<thead>
<tr>
<th></th>
<th>Pos</th>
<th>Node</th>
<th>Atom</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>NULL</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>NULL</td>
<td>“x”</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>pre(a)</td>
<td>NULL</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>NULL</td>
<td>10</td>
</tr>
</tbody>
</table>

- Problems:
  - Polymorphic columns
  - Redundant storage
  - Copy overhead (especially with strings)
Item Representation

- Item sequences, sequence order

(10, “x”, <a/>, 10) →

## Item Representation

<table>
<thead>
<tr>
<th>Pos</th>
<th>Item</th>
<th>Kind</th>
<th>Pos</th>
<th>Item</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>Int</td>
<td>1</td>
<td>1@0</td>
<td>Str</td>
</tr>
<tr>
<td>2</td>
<td>“X”</td>
<td>Str</td>
<td>2</td>
<td>1@0</td>
<td>Str</td>
</tr>
<tr>
<td>3</td>
<td>pre(a)</td>
<td>Node</td>
<td>3</td>
<td>0@0</td>
<td>Node</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>Int</td>
<td>4</td>
<td>1@0</td>
<td>Int</td>
</tr>
</tbody>
</table>

```sql
str_values
1@0  "x"
```

```sql
int_values
1@0  10
```
Iterations

- XQuery Core has been designed around the for iteration primitive:

  \[
  \text{XQuery iteration}
  \]

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

  \[
  \equiv
  
  (e[x_1/\$v], e[x_2/\$v], \ldots, e[x_n/\$v])
  \]

- Representation of \((x_1, x_2, \ldots, x_n)\):

<table>
<thead>
<tr>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>(x_2)</td>
</tr>
<tr>
<td>\ldots</td>
<td>\ldots</td>
</tr>
<tr>
<td>(n)</td>
<td>(x_n)</td>
</tr>
</tbody>
</table>

- Derive \(\$v\) as follows:
Iterations

- XQuery Core has been designed around the for iteration primitive:

\[
\text{XQuery iteration }
\]

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

\[
\equiv
\]

\[
(e[X_1/$v$], e[X_2/$v$], \ldots, e[X_n/$v$])
\]

- Representation of \((x_1, x_2, \ldots, x_n)\):

<table>
<thead>
<tr>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>(x_2)</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>(n)</td>
<td>(x_n)</td>
</tr>
</tbody>
</table>

- Derive \$v\) as follows:

<table>
<thead>
<tr>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>(x_2)</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>(n)</td>
<td>(x_n)</td>
</tr>
</tbody>
</table>
Iterations

- XQuery Core has been designed around the `for iteration` primitive:

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

- Representation of \((x_1, x_2, \ldots, x_n)\):

<table>
<thead>
<tr>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>(x_2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(n)</td>
<td>(x_n)</td>
</tr>
</tbody>
</table>

- Derive \(\$v\) as follows:

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>(x_2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(n)</td>
<td>(n)</td>
<td>(x_n)</td>
</tr>
</tbody>
</table>
Iterations

- XQuery Core has been designed around the `for` iteration primitive:

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

- Representation of \((x_1, x_2, \ldots, x_n)\):

\[
\begin{array}{|c|c|}
\hline
\text{pos} & \text{item} \\
\hline
1 & x_1 \\
2 & x_2 \\
\vdots & \vdots \\
n & x_n \\
\hline
\end{array}
\]

- Derive \(\$v\) as follows:

\[
\begin{array}{|c|c|c|}
\hline
\text{iter} & \text{pos} & \text{item} \\
\hline
1 & 1 & x_1 \\
2 & 1 & x_2 \\
\vdots & \vdots & \vdots \\
n & 1 & x_n \\
\hline
\end{array}
\]
**Iterations**

- XQuery Core has been designed around the for *iteration* primitive:

  \[
  \text{for } \nu \text{ in } (x_1, x_2, \ldots, x_n) \text{ return } e = (e[x_1/\nu], e[x_2/\nu], \ldots, e[x_n/\nu])
  \]

- Representation of \((x_1, x_2, \ldots, x_n)\):  

<table>
<thead>
<tr>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>(x_2)</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>(n)</td>
<td>(x_n)</td>
</tr>
</tbody>
</table>

- Derive \(\nu\) as follows:

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>(x_1)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>(x_2)</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>(n)</td>
<td>1</td>
<td>(x_n)</td>
</tr>
</tbody>
</table>
Loop-Lifting

- Subexpressions are compiled in dependence of iteration scope $s$ in which they appear—represented as unary relation $loop(s)$

XQuery iteration

\[
\begin{array}{c}
\text{for } \nu \text{ in } (x_1, x_2, \ldots, x_n) \\
\end{array}
\]

\[
\begin{array}{c}
s_0 \\
\end{array}
\begin{array}{c}
s_1 \\
\text{return e}
\end{array}
\]

- Single item "a" in scope $s_1$:

\[
loop(s_1) \times \begin{array}{c|c}
\text{pos} & \text{item} \\
1 & \text{"a"}
\end{array}
\]
Loop-Lifting

- Subexpressions are compiled in dependence of iteration scope $s$ in which they appear—represented as unary relation $loop(s)$

XQuery iteration

```
$0 = for $v in (x_1, x_2, \ldots, x_n)

\quad s_0

\quad s_1 [\text{return } e]
```

$loop(s_0)$

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>$n$</td>
<td>1</td>
<td>&quot;a&quot;</td>
</tr>
</tbody>
</table>

$loop(s_1)$

<table>
<thead>
<tr>
<th>iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>\vdots</td>
</tr>
<tr>
<td>$n$</td>
</tr>
</tbody>
</table>
Loop-Lifting

- Subexpressions are compiled in dependence of **iteration scope** $s$ in which they appear—represented as unary relation $\text{loop}(s)$

\[
\begin{align*}
\text{XQuery iteration} & \quad \text{loop}(s_0) & \quad \text{loop}(s_1) \\
S_0 & \quad \begin{cases} 
\text{for } v \text{ in } (x_1, x_2, \ldots, x_n) \\
S_1 & \quad \text{return } e 
\end{cases} & \\
& \quad \begin{array}{c}
\text{iter} \\
1 \\
\vdots \\
1 \\
n 
\end{array} & \quad \begin{array}{c}
\text{iter} \\
1 \\
\vdots \\
n 
\end{array}
\end{align*}
\]

- Single item "a" in scope $s_1$:  
  \[
  \begin{array}{c|c|c}
  \text{iter} & \text{pos} & \text{item} \\
  1 & 1 & \text{"a"} \\
  \vdots & \vdots & \vdots \\
n & 1 & \text{"a"}
  \end{array}
  \]

- Sequence ("a", "b") in scope $s_1$:  
  \[
  \text{loop}(s_1) \times \begin{array}{c|c}
  \text{pos} & \text{item} \\
  1 & \text{"a"} \\
  2 & \text{"b"}
  \end{array}
  \]
Loop-Lifting

- Subexpressions are compiled in dependence of **iteration scope** $s$ in which they appear—represented as unary relation $\text{loop}(s)$

**XQuery iteration**

\[
\begin{align*}
  s_0 & : \quad \text{for } \nu \text{ in } (x_1, x_2, \ldots, x_n) \\
  s_1 & : \quad \text{return } e
\end{align*}
\]

$\text{loop}(s_0)$

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>$n$</td>
<td>1</td>
<td>&quot;a&quot;</td>
</tr>
</tbody>
</table>

$\text{loop}(s_1)$

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>&quot;b&quot;</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>$n$</td>
<td>1</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>$n$</td>
<td>2</td>
<td>&quot;b&quot;</td>
</tr>
</tbody>
</table>

$\triangleright$ Single item "a" in scope $s_1$:  
$\triangleright$ Sequence ("a", "b") in scope $s_1$: 
Deriving \textit{loop}

\textbf{XQuery FLWOR expressions define a new \textit{loop} relation.}

\begin{align*}
\text{for } \nu \text{ in } (x_1, x_2, \ldots, x_n) \text{ return } e
\end{align*}

- How can we derive \textit{loop} given this XQuery expression?
- \textit{(x_1, x_2, \ldots, x_n)}
- Derive $\nu$:

\begin{tabular}{|c|c|}
\hline
\textit{pos} & \textit{item} \\
\hline
1 & $x_1$ \\
2 & $x_2$ \\
$\vdots$ & $\vdots$ \\
n & $x_n$ \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|}
\hline
\textit{iter} & \textit{pos} & \textit{item} \\
\hline
1 & 1 & $x_1$ \\
2 & 1 & $x_2$ \\
$\vdots$ & $\vdots$ & $\vdots$ \\
n & 1 & $x_n$ \\
\hline
\end{tabular}

- \textit{loop(s_1)}
Nested Scopes

Nested for blocks

\( s_0 \)
\[
\text{for } \nu_0 \text{ in } (10, 20)
\]
\[
\begin{array}{c}
\text{\( s_1 \)} \\
\text{\text{for } } \nu_1 \text{ in } (100, 200) \\
\text{\( s_2 \)} \\
\text{return } \nu_0 + \nu_1
\end{array}
\]

- Derive \( \nu_0, \nu_1 \)

\( \nu_0 \text{ in } s_1: \)

<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>2</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

\( \text{loop}(s_0) \)  
- \( \text{iter} \) 1

\( \text{loop}(s_1) \)  
- \( \text{iter} \) 1 2

\( \text{loop}(s_2) \)  
- \( \text{iter} \) 1 2 3 4
Nested Scopes

Nested for blocks

\[ s_0 \left[ \text{for } \nu_0 \text{ in (10,20) } \right. \]
\[ s_1 \left[ \text{for } \nu_1 \text{ in (100,200) } \right. \]
\[ s_2 \left[ \text{return } \nu_0 + \nu_1 \right) \]

\[ \text{loop}(s_0) \]
\[ \text{iter} \]
\[ 1 \]

\[ \text{loop}(s_1) \]
\[ \text{iter} \]
\[ \frac{1}{2} \]
\[ \frac{2}{3} \]
\[ \frac{3}{4} \]

• Derive $\nu_0, \nu_1$

$\nu_0 \text{ in } s_1:$

<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>2</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

$\nu_1 \text{ in } s_2:$

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>200</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>200</td>
</tr>
</tbody>
</table>
Loop-lifting

**Nested for blocks**

\[
\begin{align*}
  s_0 & \quad \text{for } v_0 \text{ in } (10, 20) \\
  s_1 & \quad \text{for } v_1 \text{ in } (100, 200) \\
  s_2 & \quad \text{return } v_0 + v_1
\end{align*}
\]

- Relation *map* captures the semantics of nested iteration:

\[
\begin{array}{c|c|c}
\text{inner} & 1 & 2 \\
\text{outer} & 1 & 2 \\
\end{array}
\]

- Representation of $v_0$ in $s_2$:

\[
\pi_{iter:inner, pos, item}(v_0 \bowtie_{iter=outer} \text{map}) = 
\begin{array}{c|c|c}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & 10 \\
2 & 1 & 10 \\
3 & 1 & 20 \\
4 & 1 & 20 \\
\end{array}
\]
for $v_0$ in (10, 20)
  for $v_1$ in (100, 200)
  s2[
    return $v_0 + v_1$
  ]
### XQuery Construct
- sequence construction
- if-then-else
- for-loops
- calculations
- list functions, e.g. `fn:first()`
- element construction
- XPath steps

### Relational Mapping
- A union B
- `select(A=true,B) union select(A=false,C)`
- cartesian product
- `project(A,x=expr(Y1,..Yn))`
- `select(A,pos=1)`
- updates in temporary tables
- staircase-join
XMark Query 8

```xquery
let $auction := doc("auctions.xml")
return
for $p in $auction/site/people/person
let $a :=
  for $t in $auction/site/
closed_auctions/closed_auction
where $t/buyer/@person = $p/@id
return $t
return
$item person="{$p/name/text()}">
  {count($a)}
</item>
```
Peephole Optimization
[Grust, XIME-P 2005]

Input: XQuery

for $x$ in (k, ..., 2, 1)
return $x * 5$

Output: Relational Algebra

![Diagram showing relational algebra with iter, pos, item columns and operations like projection, join, and multiplication.]
Peephole Optimization

[Grust, XIME-P 2005]

Plan Property: Constant Columns

Input: XQuery

for $x$ in (k, ..., 2, 1)
return $x * 5$
Peephole Optimization
[Grust, XIME-P 2005]

Plan Property: Strictly Required Columns

\[ \pi_{\text{iter}:1, \text{pos}:\text{pos1}, \text{item}} \]
\[ \epsilon_{\text{pos1}:\text{iter}} \]
\[ \pi_{\text{outer}:1, \text{inner}} \]
\[ \pi_{\text{iter}, \text{pos}:1, \text{item}:\text{res}} \]
\[ \odot_{\text{res}:\text{item},5} \]
\[ \pi_{\text{iter}=\text{iter1}} \]
\[ \pi_{\text{iter1}, \text{iter}, \text{item}:5} \]
\[ \pi_{\text{iter}} \]
\[ \pi_{\text{iter}, \text{inner}, \text{item}} \]
\[ \epsilon_{\text{inner}:\text{pos}} \]

\begin{array}{ccc}
\text{iter} & \text{pos} & \text{item} \\
1 & 1 & k \\
\vdots & \vdots & \vdots \\
1 & k & 1 \\
\end{array}

Input: XQuery

\[
\text{for } x \text{ in } (k, \ldots, 2, 1) \\
\text{return } x \times 5
\]
Peephole Optimization
[Grust, XIME-P 2005]

Plan Property: Key Candidate Columns

Input: XQuery

```xquery
for $x in (k, ..., 2, 1)
return $x * 5
```
Peephole Optimization
[Grust, XIME-P 2005]

Plan Property: Dense Columns

\[ \begin{align*}
\pi_{\text{iter},1}, & \text{pos}:\text{pos1},item \\
Q_{\text{pos1}:}\text{(iter)} & \\
\pi_{\text{iter},\text{item},\text{inner}:\text{iter}} & \\
\pi_{\text{iter},\text{item}:\text{res}} & \\
\otimes_{\text{res}:}\text{(item,5)} & \\
\pi_{\text{iter},\text{item},\text{iter1}:\text{iter}} & \\
\pi_{\text{iter}:\text{inner},\text{item}} & \\
Q_{\text{inner}:}\text{(pos)} & \\
\end{align*} \]

\[\text{dense: iter, pos} \]

Input: XQuery

\[
\text{for } x \text{ in } (k, \ldots, 2, 1) \\text{ return } x \times 5
\]
Peephole Optimization
[Grust, XIME-P 2005]

Input: XQuery

```
for $x in (k, ..., 2, 1)
  return $x * 5
```

Final Plan

```
π_{iter:1,pos,item:res}
⊗_{res:(item,5)}
```

<table>
<thead>
<tr>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>k</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>1</td>
</tr>
</tbody>
</table>
Peephole Optimization
[Grust, XIME-P 2005]

• Plan simplification

Plan Property: Constant Columns

Final Plan

Input: XQuery
for $x$ in (k, ..., 2, 1)
return $x \times 5$
Peephole Optimization
[Boncz et al., SIGMOD 2006]

• Sort Avoidance

generated by DENSE RANK pos ORDER BY pos PARTITION BY iter
by tracking secondary ordering properties [pos|iter], [Wang&Cherniack, VLDB’03]

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>Z</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>C</td>
</tr>
</tbody>
</table>

[iter,pos]
Peephole Optimization
[Boncz et al., SIGMOD 2006]

• Sort Avoidance

generated by DENSE RANK pos ORDER BY pos PARTITION BY iter

by tracking secondary ordering properties [pos|iter], [Wang&Cherniack, VLDB’03]
Peephole Optimization
[Boncz et al., SIGMOD 2006]

Sort Avoidance

generated by DENSE RANK pos ORDER BY pos PARTITION BY iter
by tracking secondary ordering properties [pos|iter], [Wang&Cherniack, VLDB’03]

⇒ hash-based (streaming) DENSE_RANK

<table>
<thead>
<tr>
<th>iter</th>
<th>pos</th>
<th>item</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>Z</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>C</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>X</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>C</td>
</tr>
</tbody>
</table>

[pos|iter]
**Peephole Optimization**

[Boncz et al., SIGMOD 2006]

- **Sort Avoidance**

  generated by `DENSE RANK pos ORDER BY pos PARTITION BY iter`

  by tracking secondary ordering properties `[pos|iter]`, [Wang&Cherniack, VLDB’03]

  ➔ hash-based (streaming) `DENSE_RANK`

---

**Graph:**

- **Normalized performance/speedup**
  - non-order preserving
  - order preserving

**Legend:**

- **XMark, 110MB**

**Axes:**

- **Q:** 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
Peephole Optimization
[Boncz et al., SIGMOD 2006]

• Sort Avoidance

generated by DENSE RANK pos ORDER BY pos PARTITION BY iter
by tracking secondary ordering properties [pos|iter], [Wang&Cherniack, VLDB'03]
⇒ hash-based (streaming) DENSE_RANK

• Sort Reduction

if [iter,item] order is required, and [iter] only is present
use a refine-sort rather than a full sort.
⇒ pipelinable sort
Peephole Optimization
[Boncz et al., SIGMOD 2006]

• Sort Avoidance

generated by DENSE RANK \texttt{pos ORDER BY pos PARTITION BY iter}
by tracking secondary ordering properties [\texttt{item|iter}], [Wang & Cherniack, VLDB’03]
➔ hash-based (streaming) DENSE_RANK

• Sort Reduction

if [\texttt{iter,item}] order is required, and [\texttt{iter}] only is present
use a \texttt{refine-sort} rather than a full sort.
➔ pipelinable sort

• Join Detection

detect cartesian products as multi-valued dependencies
in the presence of a theta-selection ➔ theta-join
Loop-lifted XPath Steps

Many algorithms have been proposed & studied for XPath evaluation:
• Dataguide based,
• Structural Join,
• Staircase Join,
• Holistic Twig Join

IN: sequence of context nodes in (doc order)
OUT: sequence of document nodes (unique, in doc order)
Loop-lifted XPath Steps

In XQuery, expressions generally occur inside FLWR blocks, i.e. inside a for-loop

\[
\text{for } x \text{ in } \text{doc()//employee} \\
\quad \text{$x$/ancestor::department}
\]

Choice:
- call XPath algorithm N times, accessing document and index structures N times.
- use a **loop-lifted** algorithm:

**IN:** for each iteration, a sequence of context nodes
**OUT:** for each iteration, a sequence of document nodes
(per iteration unique, in doc order)
Staircase join

document

List of context nodes
Loop-lifted staircase join

Adapt:

**pruning**, **partitioning** and **skipping** rules
to correctly deal with multiple context sets
Loop-lifted staircase join

Results on the 20 XMark queries:
Performance Evaluation

Extensive performance Evaluation on XMark
• data sizes 110KB, 1MB, 11MB, 110MB, 1.1GB, 11GB
• MonetDB/XQuery, Galax0.6, X-Hive 6.0, Berkeley DB XML 2.0, eXisT
• 8GB RAM

Extensive XMark performance Literature Overview
• IPSI-XQ v1.1.1b, Dynamic Interval Encoding, Kweelt, QuiP, FluX, TurboXPath, Timber, Qizx/Open (Version 0.4/p1), Saxon (Version 8.0), BEA/XQRL, VX
• Crude comparison (normalized by CPU SPECint)
XMark Benchmark

1MB XML

Galax

X-Hive

Berkeley DB XML

eXisT

1GB XML
More Benchmarks & Performance Results?

• http://monetdb.cwi.nl/XQuery/Benchmark/