Technische Universität München

Why Off-The-Shelf RDBMSs are Better at XPath Than You Might Expect

Jens Teubner · Torsten Grust · Jan Rittinger

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

XML Processing on Relational Back-Ends

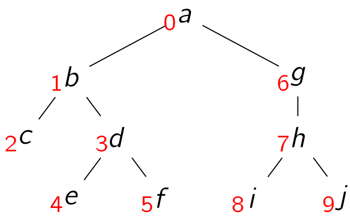
We do **not** want to clutter the RDBMS kernel with XPath specifics, e.g.,

- Multi-Predicate Merge Joins (MPMGJN),
- Holistic join algorithms (PathStack, TwigStack, etc.),
- Structural joins (Tree-Merge, Stack-Tree, staircase join, etc.).

Instead: Use **existing** functionality in **off-the-shelf** RDBMSs:

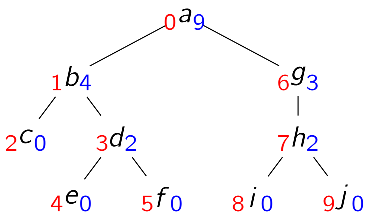
- (partitioned) **B-trees**,
- **aggregates**.

Recap: Tree Encodings



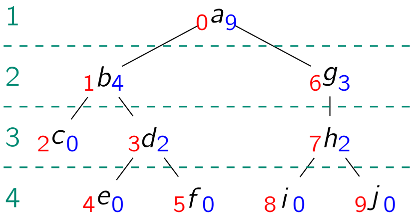
n	pre
a	0
b	1
c	2
d	3
e	4
f	5
g	6
$\vdots$	$\vdots$

Recap: Tree Encodings



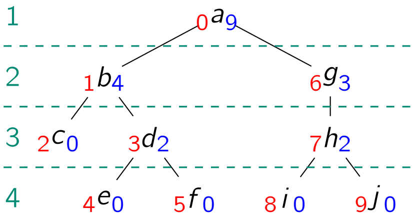
n	*pre*	*size*
a	0	9
b	1	4
c	2	0
d	3	2
e	4	0
f	5	0
g	6	3
⋮	⋮	⋮

Recap: Tree Encodings

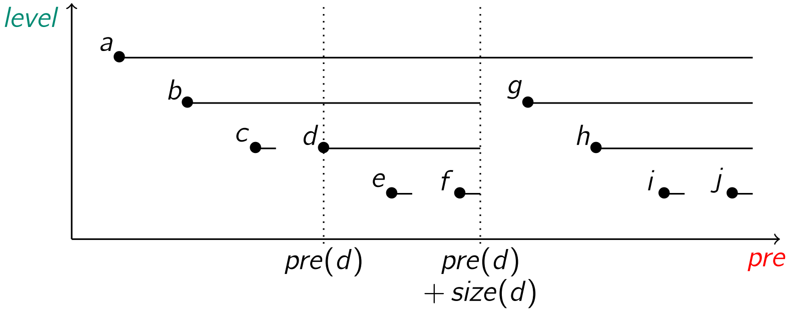


n	*pre*	*size*	*level*
a	0	9	1
b	1	4	2
c	2	0	3
d	3	2	3
e	4	0	4
f	5	0	4
g	6	3	2
⋮	⋮	⋮	⋮

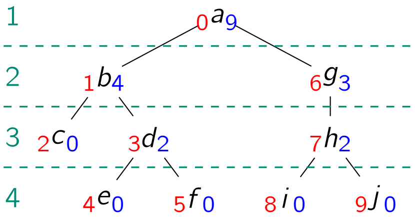
Recap: Tree Encodings



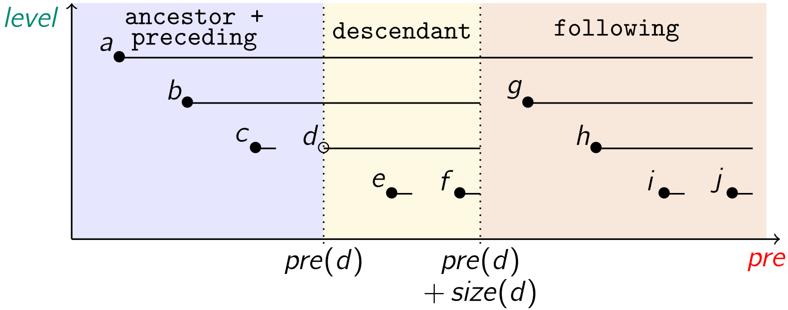
n	*pre*	*size*	*level*
a	0	9	1
b	1	4	2
c	2	0	3
d	3	2	3
e	4	0	4
f	5	0	4
g	6	3	2
⋮	⋮	⋮	⋮



Recap: Tree Encodings



n	*pre*	*size*	*level*
a	0	9	1
b	1	4	2
c	2	0	3
d	3	2	3
e	4	0	4
f	5	0	4
g	6	3	2
⋮	⋮	⋮	⋮



XPath on Encoded Tree Data

The XPath `descendant` axis turns into a **range predicate** on *pre*.

✓ Efficiently supported by a **B-tree** index on column *pre*.

XPath on Encoded Tree Data

The XPath `descendant` axis turns into a **range predicate** on *pre*.

✓ Efficiently supported by a **B-tree** index on column *pre*.

The `child` axis needs some more thought (e.g., *ctx*`/child::node()`):

```
SELECT DISTINCT d.*
  FROM ctx c, document d
 WHERE c.pre < d.pre AND d.pre ≤ c.pre + c.size
   AND d.level = c.level + 1
 ORDER BY d.pre
```

no simple range
(*level* condition)
→ **false hits**

XPath on Encoded Tree Data

The XPath `descendant` axis turns into a **range predicate** on *pre*.

✓ Efficiently supported by a **B-tree** index on column *pre*.

The `child` axis needs some more thought (e.g., *ctx*/`child::node()`):

$$
\begin{array}{l}
\texttt{SELECT DISTINCT}\ d.*\\
\quad \texttt{FROM}\ \mathrm{ctx}\ c\texttt{,}\ \mathrm{document}\ d\\
\texttt{WHERE}\ c.pre < d.pre\ \texttt{AND}\ d.pre \leq c.pre + c.size\\
\quad \texttt{AND}\ d.level = c.level + 1\\
\texttt{ORDER BY}\ d.pre
\end{array}
$$

no simple range
(*level* condition)
→ **false hits**

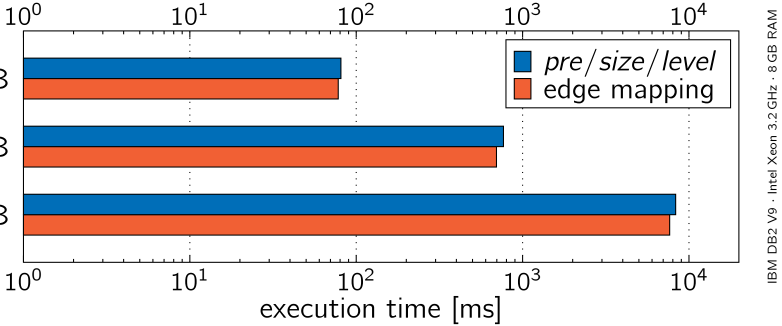
Contrast to **edge mapping** (explicit parent/child edges):

$$
\begin{array}{l}
\texttt{SELECT DISTINCT}\ d.*\\
\quad \texttt{FROM}\ \mathrm{ctx}\ c\texttt{,}\ \mathrm{document}\ d\\
\texttt{WHERE}\ c.pre = d.parent\\
\texttt{ORDER BY}\ d.pre
\end{array}
$$

foreign key join

XPath on Encoded Tree Data — Experiment

XMark: `/descendant::open_auction/bidder/increase`



pre / size / level:

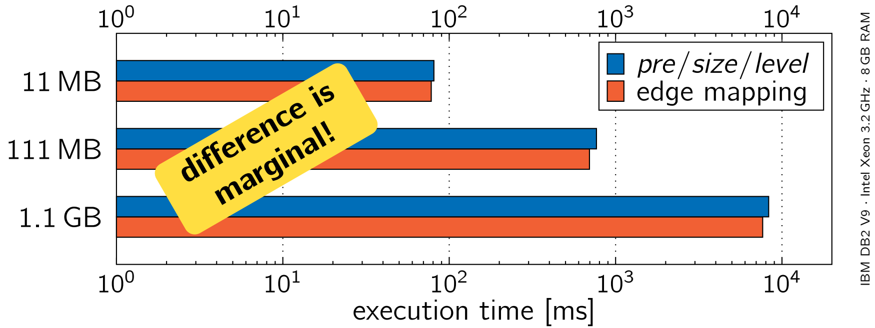
SELECT DISTINCT $d.*$
FROM ctx c, document d
WHERE $c.pre < d.pre$ AND $d.pre \leq c.pre + c.size$
AND $d.level = c.level + 1$
ORDER BY $d.pre$

edge mapping:

SELECT DISTINCT $d.*$
FROM ctx c, document d
WHERE $c.pre = d.parent$
ORDER BY $d.pre$

XPath on Encoded Tree Data — Experiment

XMark: `/descendant::open_auction/bidder/increase`



pre / size / level:

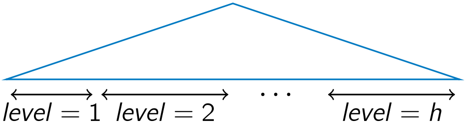
SELECT DISTINCT $d.*$
FROM ctx c, document d
WHERE $c.pre < d.pre$ AND $d.pre \leq c.pre + c.size$
AND $d.level = c.level + 1$
ORDER BY $d.pre$

edge mapping:

SELECT DISTINCT $d.*$
FROM ctx c, document d
WHERE $c.pre = d.parent$
ORDER BY $d.pre$

Partitioned B-Trees [Graefe 2003]

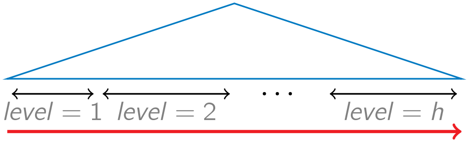
To evaluate `child`, DB2 used a $\langle \mathit{level}, \mathit{pre} \rangle$ B-tree.



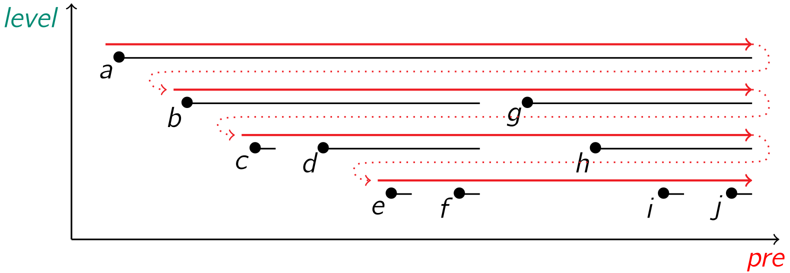
- *level* has a **low selectivity**.
- This effectively **partitions** the B-tree into h partitions (h: height of the **XML tree**).

Partitioned B-Trees [Graefe 2003]

To evaluate `child`, DB2 used a $\langle \mathit{level}, \mathit{pre} \rangle$ B-tree.

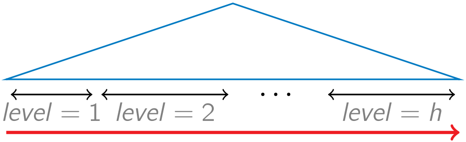


- *level* has a **low selectivity**.
- This effectively **partitions** the B-tree into h partitions (h: height of the **XML tree**).

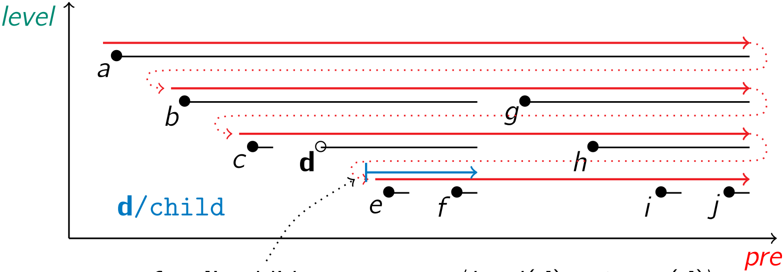


Partitioned B-Trees [Graefe 2003]

To evaluate `child`, DB2 used a $\langle \mathit{level}, \mathit{pre} \rangle$ B-tree.



- *level* has a **low selectivity**.
- This effectively **partitions** the B-tree into h partitions (h: height of the **XML tree**).



scan for **d**'s children, start at $\langle \mathit{level}(\mathbf{d}) + 1, \mathit{pre}(\mathbf{d}) \rangle$

$\rightarrow$ **no false hits!**

More Partitioned B-Trees

Use partitioned B-trees depending on your **query workload**:

- XPath **name tests**:

  $\rightarrow$ partitioned $\langle tag, pre \rangle$ or $\langle tag, level, pre \rangle$ index.

- XPath **kind tests** (e.g., `text()`, `element()`, `*`):

  $\rightarrow$ partitioned $\langle kind, pre \rangle$ or $\langle kind, level, pre \rangle$ index.

$\rightarrow$ **Predicate pushdown** into the index.

Partitioned B-trees can implement **schema-awareness**:

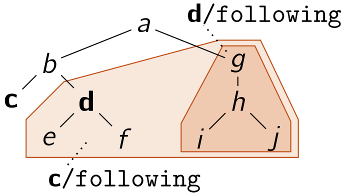
- Record root-to-leaf path for each node in column *path* ($\rightsquigarrow$ *PATH_ID* field in SQL Server's *primary XML index*).

  $\rightarrow$ partitioned $\langle path, pre \rangle$ index.

Staircase join: **prune** context nodes that won't contribute to the result.

- E.g., (**c**, **d**)/`following::node()`
- Removing **d** from the context set does not affect query outcome (XPath: duplicate-free result).

→ **Prune** context set first.

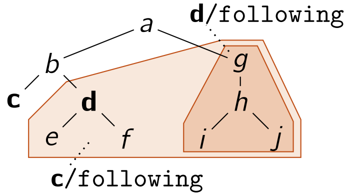


Context Pruning in an Off-The-Shelf RDBMS

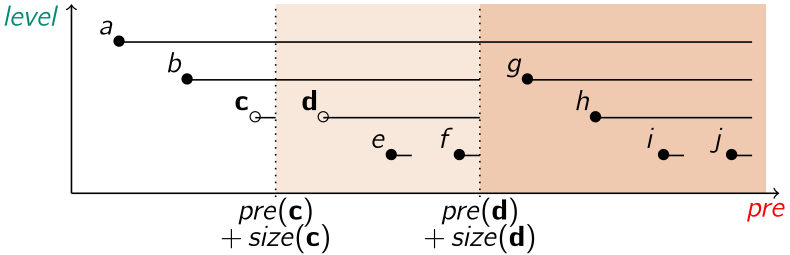
Staircase join: **prune** context nodes that won't contribute to the result.

- E.g., (**c**, **d**)`/following::node()`
- Removing **d** from the context set does not affect query outcome (XPath: duplicate-free result).

→ **Prune** context set first.



In the *pre*/*level* plane:



Context Pruning in SQL

Reduce context to node v with **minimum** $pre(v) + size(v)$.

→ Pruning turns into **aggregation** on the relational back-end.

In SQL:

```
SELECT DISTINCT d.*
  FROM ctx c, document d
 WHERE d.pre > c.pre + c.size
 ORDER BY d.pre
```

⇒

```
SELECT d.*
  FROM document d
 WHERE d.pre > ( SELECT MIN (c.pre + c.size)
                   FROM ctx c )
 ORDER BY d.pre
```

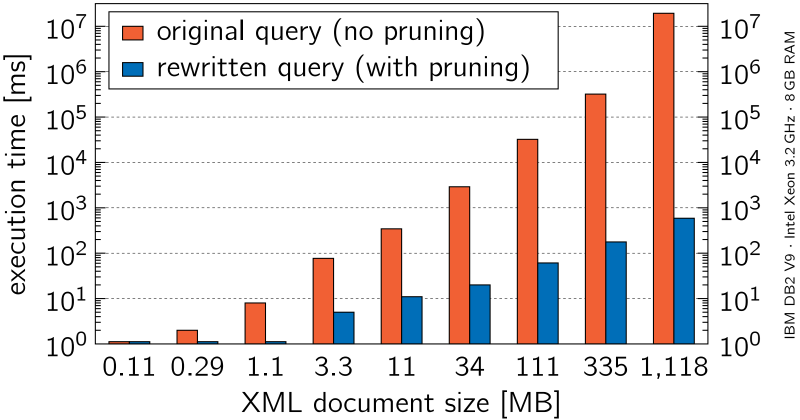
Note that this can be done by **purely algebraic rewrites**.

→ **No** XML/tree knowledge involved.

→ Also **non-XPath** queries may benefit from such rewrites.

Context Pruning on IBM DB2

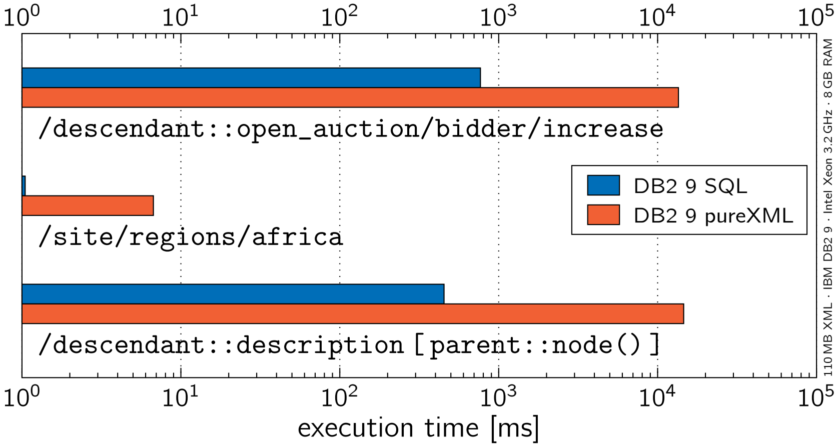
Path: `/descendant::city/following::zipcode`



Without pruning: 8.1×10^9 duplicates to **sort** on 1.1 GB XML instance!

Relational vs. Native XML

Relational vs. DB2's built-in **native** (pureXML®) XML storage.



Conclusions

- **Off-the-shelf** RDBMSs provide everything we need for efficient XML processing:
  - → **Partitioned B-trees** support non-recursive axes and others.
  - → **Aggregation** implements the **pruning** idea of staircase join.
- **Relational** XPath evaluation can outperform **state-of-the-art native** XML processors.

- The **Pathfinder** XQuery compiler exploits these ideas in its upcoming **SQL code generator**.
  - → `http://www.pathfinder-xquery.org/`
  - → Demo session 4, tomorrow afternoon

Pathfinder is supported by the German Research Foundation DFG.