

XQuery Join Graph Isolation

(Celebrating 30+ years of XQuery processing technology)



What's going on?

This work describes technology that turns an off-the-shelf relational database system into an XQuery processor. Nested expressions of the fully compositional XQuery language are initially compiled into stacked algebraic plans.

Current commercial SQL query optimizers are overwhelmed by the stacked plans that feature scattered distribution of join, duplicate elimination, and ordering. Our approach provides a set of rewrite rules that extract duplicate elimination and ordering from the stacked plans and thus separate a join graph from the blocking operators in the plan tail.

Translating the join graph plans into SQL lets the relational database query optimizer face a problem known inside out. In our experiments we have observed the IBM DB2 query optimizer to autonomously "reinvent" advanced processing strategies that have originally been devised specifically for the XQuery domain.

Source: XQuery

```

Expr → for $VarName in Expr return Expr
      | $VarName
      | if (BoolExpr) then Expr else ()
      | doc(StringLiteral)
      | Expr / ForwardAxis NodeTest
      | Expr / ReverseAxis NodeTest
BoolExpr → Expr
GeneralComp → Expr GeneralComp Literal
ForwardAxis → descendant:: | following:: | ...
ReverseAxis → parent:: | ancestor:: | ...
NodeTest → KindTest | NameTest
Literal → NumericLiteral | StringLiteral
VarName → QName
StringLiteral → "... "
    
```

Intermediate: Algebra

We compile XQuery expressions into standard table algebra plans consisting of table references (Θ), selection (σ), projection (π), join (\bowtie), cross product (\times), duplicate elimination (δ), and ranking/numbering operators (ρ , #).

Join Graph Isolation

A compile-time data flow analysis derives for each operator (1) constant, (2) key, (3) (absence of) duplicate, and (4) utilization property information.

A peep-hole style optimizer rewrites small patterns (consisting of at most two operators) based on the collected property information.

The rewrite rules pursue three goals:

1. establish a single ρ operator in the plan tail,
2. push-down and remove \bowtie operators, and
3. establish a single δ operator in the plan tail.

```

doc("auction.xml")
/descendant::open_auction[bidder]
    
```

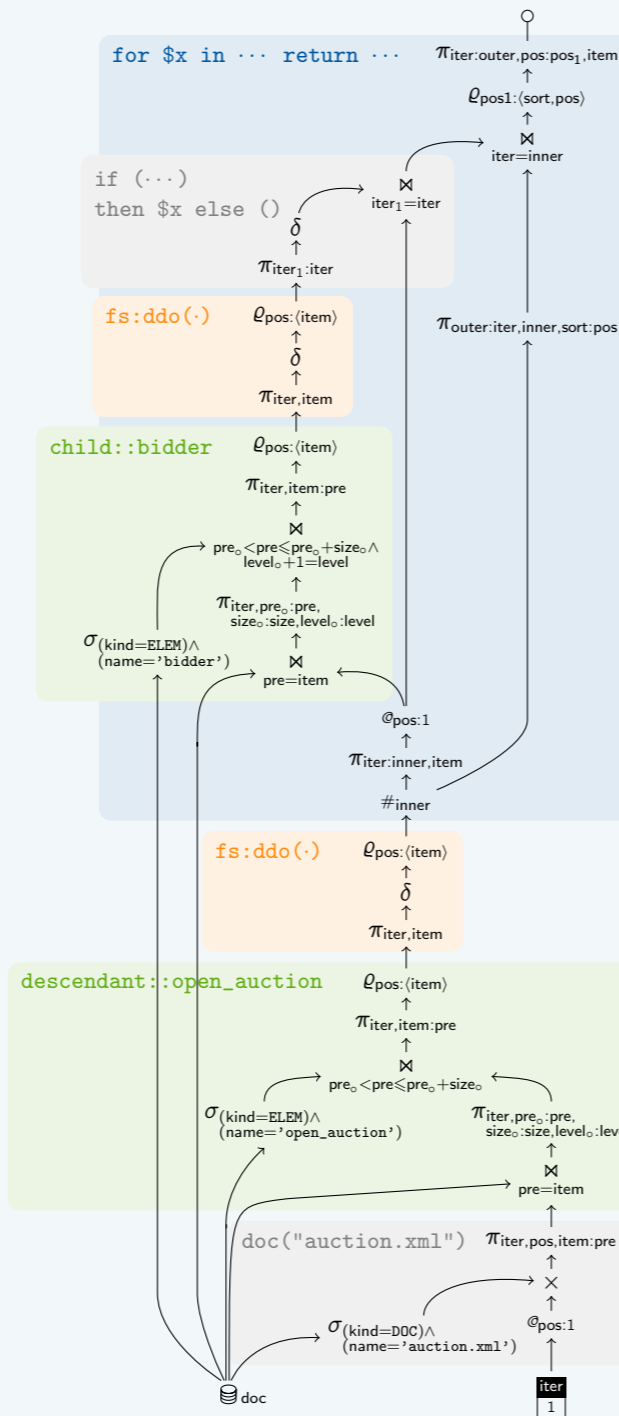
```

for $x in fs:ddo(doc("auction.xml"))
  /descendant::open_auction
return if (fn:boolean(fs:ddo($x/child::bidder)))
  then $x else ()
    
```

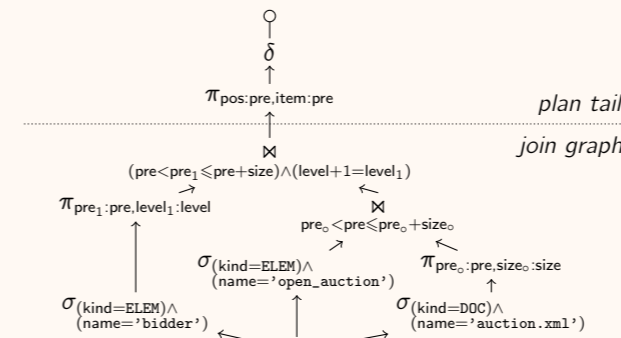
```

<XQueryResult><open_auction id="open_auction0"><initial>109.51</initial><bidder
<date>05/26/2001</date><time>16:06:53</time><personref person="person103"/><incr
ease>27.00</increase></bidder><bidder><date>08/16/2000</date><time>17:58:31</tim
e><personref person="person102"/><increase>6.00</increase></bidder><bidder><date
>09/06/2000</date><time>21:45:20</time><personref person="person66"/><increase>1
0.50</increase></bidder><bidder><date>12/08/2000</date><time>15:04:47</time><per
sonref person="person94"/><increase>31.50</increase></bidder><bidder><date>05/14
/2000</date><time>18:55:55</time><personref person="person91"/><increase>21.00<
/increase></bidder><bidder><date>02/11/1999</date><time>11:00:00</time><personr
ef person="person94"/><increase>12.00</increase></bidder><bidder><date>02/13/1998
</date><time>19:16:35</time><personref person="person13"/><increase>6.00</increa
se></bidder><bidder><date>09/17/2000</date><time>06:19:57</time><personref perso
n="person72"/><increase>6.00</increase></bidder><bidder><date>11/05/2001</date><
time>11:41:20</time><personref person="person33"/><increase>15.00</increase></bi
dder><bidder><date>03/07/2001</date><time>02:09:44</time><personref person="pers
on92"/><increase>19.50</increase></bidder><bidder><date>09/18/2001</date><time>0
    
```

Stacked Plan



Join Graph

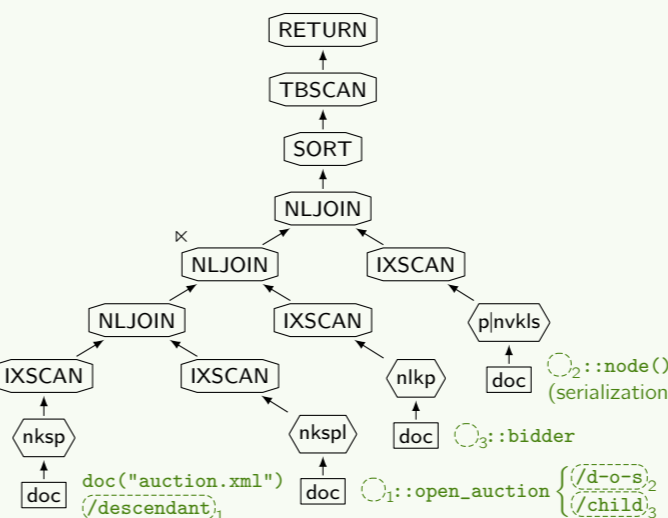


Target: SQL Code

```

SELECT DISTINCT d4.*, d2.pre as dist
FROM doc AS d1, doc AS d2, doc AS d3, doc AS d4
WHERE d1.kind = DOC AND d1.name = 'auction.xml'
AND d2.kind = ELEM AND d2.name = 'open_auction'
AND d3.kind = ELEM AND d3.name = 'bidder'
AND d2.pre BETWEEN d1.pre + 1 AND d1.pre + d1.size
AND d3.pre BETWEEN d2.pre + 1 AND d2.pre + d2.size
AND d2.level + 1 = d3.level
AND d4.pre BETWEEN d2.pre + 1 AND d2.pre + d2.size
ORDER BY d2.pre, d4.pre
    
```

IBM DB2 Execution Plan



IBM DB2 Observations

Based on isolated join graphs DB2 is autonomously able to

- detect a good set of partitioned B-Tree indexes,
- exploit B-Tree indexes like element tag streams,
- use B-Tree indexes like XPath-specific indexes,
- apply step reversal or even hybrid path evaluation,
- detect early-out semantics for predicates (\times), and
- beat its pureXML™ equivalent for larger documents.

Queries

Query	Data
Q ₁ /descendant::open_auction[bidder]	XMark
Q ₂ for \$a in //closed_auction[price>500], \$i in //item, \$c in //category where \$a/itemref/@item = \$i/@id and \$i/incategory/@category = \$c/@id return \$c/name	XMark
Q ₃ /site/people/person[@id = "person0"] /name/text()	XMark
Q ₄ //closed_auction/price/text()	XMark
Q ₅ /dblp/*[@key = "conf/vldb2001" and editor and title]/title	DBLP
Q ₆ for \$thesis in /dblp/phdthesis [year < "1994" and author and title] return-tuple \$thesis/title, \$thesis/author, \$thesis/year	DBLP

Measurements

Query	# nodes	DB2 + Pathfinder stacked join graph		DB2 pureXML™ whole segmented	
		🕒 (sec)	🕒 (sec)	🕒 (sec)	🕒 (sec)
Q ₁	1,625,157	63.011	11.788	10.073	9.661
Q ₂	318	DNF	0.544	DNF	DNF
Q ₃	1	60.582	0.017	0.891	0.001
Q ₄	9,750	32.246	0.309	6.455	7.438
Q ₅	1	442.745	0.391	48.066	0.001
Q ₆	59	0.026	0.004	1.292	0.017

Visit us at www.pathfinder-xquery.org



Torsten Grust torsten.grust@uni-tuebingen.de
 Manuel Mayr manuel.mayr@uni-tuebingen.de
 Jan Rittinger jan.rittinger@uni-tuebingen.de