XML and Databases

Lecture 10
XPath Evaluation using RDBMS

Sebastian Maneth
NICTA and UNSW

CSE@UNSW -- Semester 1, 2009

Outline

1. Recall pre / post encoding
2. XPath with //, ancestor, @, and text()
3. XPath with / and following-sibling:
   → use pre / size / level encoding
4. Optimization through tree-aware join: "*Staircase Join*"
   → Prune
   → Partition and Skip

Outline - Lectures

1. Introduction to XML, Encodings, Parsers
2. Memory Representations for XML: Space vs Access Speed
3. RDBMS Representation of XML
4. DTDs, Schemas, Regular Expressions, Ambiguity
5. XML Validation using Automata

XPath

6. Node Selecting Queries: **XPath**
7. Tree Automata for Efficient **XPath** Evaluation, Parallel Evaluation
8. .**XPath** Properties: backward axes, containment test
9. Streaming Evaluation: how much memory do you need?
10. **XPath** Evaluation using RDBMS

11. XSLT – stylesheets and transform
12. XQuery – XML query language

Outline - Assignments

1. Read XML, using DOM parser. Create document statistics.
2. SAX Parse into memory structure: Tree and DAG
3. Map XML into
4. XPath evaluation
5. **XPath** into SQL Translation → 1. June

… from Lecture 3 :

Questions

What is the benefit of storing the LEVEL of a node in the table?
Which accessor functions become easier?

→ It can also be useful to store the *size of the subtree* at a node in the table.
Do you see advantages of doing that?

… from Lecture 3 :

Later in this course, we will use the PRE/POST encoding again.

→ We will find a systematic way to *map* queries on XML (XPath) into XQL queries.

Assignment 5 is about programming this mapping.

Assignment 3 generate tables, send to DB, run query & print results

XML document

```
<book isbn="1-2345-6789-0" year="1994">
<title>TCP/IP Illustrated</title>
<author><last>Stevens</last><first>John</first></author>
<publisher>Addison-Wesley</publisher>
<price currency="USD">65.95</price>
</book>
```

book_tbl:

```
1  12   book      null
2  2    title     null
3  1    null      TCP/IP Illustrated
4  7    author    null
5  4    last      null
6  3    null      Stevens
7  6    first     null
8  5    null      John
9  9    publisher       null
10 8    null      Addison-Wesley
11 11   price     null
12 10   null      65.95
```

book_attr:

```
1 isbn 1-2345-6789-0
1 year 1994
11 currency USD
```

7

Assignment 3 generate tables, send to DB, run query & print results

XML document

```
<book isbn="1-2345-6789-0" year="1994">
<title>TCP/IP Illustrated</title>
<author><last>Stevens</last><first>John</first></author>
<publisher>Addison-Wesley</publisher>
<price currency="USD">65.95</price>
</book>
```

book_tbl:

```
1  12   book      null
2  2    title     null
3  1    null      TCP/IP Illustrated
4  7    author    null
5  4    last      null
6  3    null      Stevens
7  6    first     null
8  5    null      John
9  9    publisher       null
10 8    null      Addison-Wesley
11 11   price     null
12 10   null      65.95
```

book_attr:

```
1 isbn 1-2345-6789-0
1 year 1994
11 currency USD
```

Assignment 5

add extra root node (pre=0)

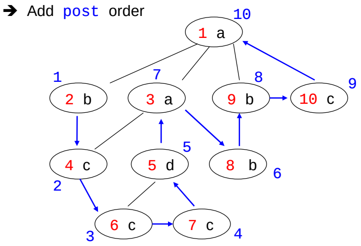
→ Generate SQL queries from Xpath
(1) //, node tests: * and "a", etc
(2) add / and ancestor
(3) add filters

Bonus: add @-axis

8

1. Recall: Pre/Post Encoding

→ Add post order



pre	post	lab
1	10	a
2	1	b
3	7	a
4	2	c
5	5	d
6	3	c
7	4	c
8	6	b
9	8	b
10	9	c

```
CREATE VIEW descendant AS
SELECT r1.pre,r2.pre FROM doc_tbl r1, R r2
          WHERE r1.pre<r2.pre
            AND r1.post>r2.post
```

"structural join"

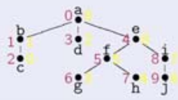
9

XPath Accelerator Encoding | Pre-Order and Post-Order Traversal Ranks

XPath Accelerator encoding

XML fragment *f* and its skeleton tree

```
<a>
  <b>c</b>
  <!--d-->
  <e><f><g/><?h?></f>
     <i>j</i>
  </e>
</a>
```



Pre/post encoding of *f*: table accel

pre	*post*	*par*	*kind*	*tag*	*text*
0	9	NULL	elem	a	NULL
1	1	0	elem	b	NULL
2	0	1	text	NULL	c
3	2	0	com	NULL	d
4	8	0	elem	e	NULL
5	5	4	elem	f	NULL
6	3	5	elem	g	NULL
7	4	5	pi	NULL	h
8	7	4	elem	i	NULL
9	6	8	text	NULL	j

Marc H. Scholl (DBIS, Uni KN) XML and Databases Winter 2005/06 353

XML & DATABASES

XML Database – Table Storage

Attributes:

- attributes are "owned" by their elements → no table storage
- another table field is introduced which references attribute arrays

PRE	POST	LEVEL	TYPE	TOKEN	ATTS
1	15	1	elem	html	
2	3	2	elem	head	
3	2	3	elem	title	
4	1	4	text	XML	
5	14	2	elem	body	atts
6	5	3	elem	h1	
7	4	4	text	Databases...	
8	13	3	elem	div	atts
9	7	4	elem	b	
...	...	...	...	...	

ATTRIBUTE	VALUE
bgcolor	#FFFFFF
text	#000000

ATTRIBUTE	VALUE
align	right

11

Our Tables: Assignment 3 + extra root node

XML document

```
<book isbn="1-2345-6789-0" year="1994">
<title>TCP/IP Illustrated</title>
<author><last>Stevens</last><first>John</first></author>
<publisher>Addison-Wesley</publisher>
<price currency="USD">65.95</price>
</book>
```

book_tbl:

```
0  13   0      null      null
1  12   1      book      null
2  2    2      title     null
3  1    3      null      TCP/IP Illustrated
4  7    2      author    null
5  4    3      last      null
6  3    4      null      Stevens
7  6    3      first     null
8  5    4      null      John
9  9    2      publisher         null
10 8    3      null      Addison-Wesley
11 11   2      price     null
12 10   3      null      65.95
```

book_attr:

```
1 isbn 1-2345-6789-0
1 year 1994
11 currency USD
```

Add root node with
→ pre = 0
→ post = #nodes+1
→ tag = null
→ text = null

12

Our Tables: Assignment 3 + extra root node

XML document

```
<book isbn="1-2345-6789-0" year="1994">
<title>TCP/IP Illustrated</title>
<author><last>Stevens</last><first>John</first></author>
<publisher>Addison-Wesley</publisher>
<price currency="USD">65.95</price>
</book>
```

book_tbl:

0	13	0	null	null
1	12	1	book	null
2	2	2	title	null
3	1	3	null	TCP/IP Illustrated
4	7	2	author	null
5	4	3	last	null
6	3	4	null	Stevens
7	6	3	first	null
8	5	4	null	John
9	9	2	publisher	null
10	8	3	null	Addison-Wesley
11	11	2	price	null
12	10	3	null	65.95

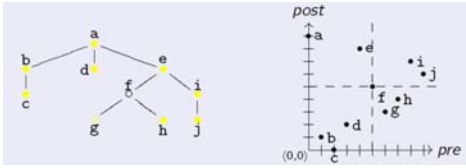
book_attr:

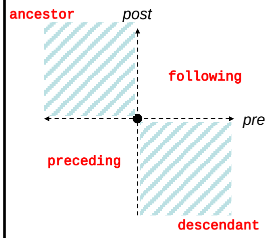
1	isbn	1-2345-6789-0
1	year	1994
11	currency	USD

Add root node with
- → pre = 0
- → post = #nodes+1
- → tag = null
- → text = null

More flexible, of course, to use TYPE-field with values { elem, text, root, com, PI .. }

13





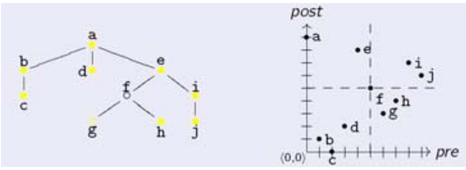
descendant(pr, po) =

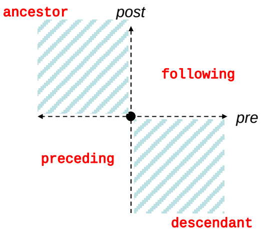
```
SELECT r1.pre FROM doc_tbl r1
       WHERE r1.pre>pr
         AND r1.post<po
     ORDERED BY r1.pre
```

ancestor(pr, po) =

```
SELECT r1.pre FROM doc_tbl r1
       WHERE r1.pre<pr
         AND r1.post>po
     ORDERED BY r1.pre
```

14





descendant(pr, po) =

```
SELECT r1.pre FROM doc_tbl r1
       WHERE r1.pre>pr
         AND r1.post<po
     ORDERED BY r1.pre
```

XPath evaluation: need to operate wrt a **set of context nodes**.

stored in *intermediate table*

15

XPath evaluation (approach)

Universität Konstanz University of Twente The Netherlands

XPath path expression $s_1/s_2/\ldots/s_n$
- Each step s_i is of the form *axis::nodetest*
- Each step s_i results in **context node sequence** for step s_{i+1}

Evaluation in DBMS
- ➢ Apply each location step to *all* nodes in context (bulk-oriented query processing)
- ➢ **Preserve document order** and **not produce duplicate nodes**

Initial focus on *major axis steps*
- Major: *ancestor*, *descendant*, *preceding*, and *following*
- Other axes define efficiently computable subsets of the four major axes

6 september 2004 EDBT summerschool - Fully (or Purely) Relational XPath and XQuery Processors 14

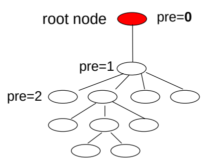
2. XPath with //, ancestor, @, and text()

/book//author[./text()="Knuth"]

location step **1**: "root node"

#location steps

```
SELECT r3.pre FROM book_tbl r1, r2, r3
       WHERE r1.pre=0
       AND
```



Watch out:

This must be "extra" root node
→ *XPath data model*

17

2. XPath with //, ancestor, @, and text()

//book//author[./text()="Knuth"]

location step **2**: "descendant :: book"

```
SELECT r3.pre FROM book_tbl r1, r2, r3
       WHERE r1.pre=0
       AND r2.pre>r1.pre
       AND r2.post<r1.post
       AND r2.tag="book"
       AND
```

Formally: do you remember the correct definition of the abbreviation "//"?

// is abbreviation for /descendant-or-self::node()/

//book is abbreviation for $\underbrace{\text{/descendant-or-self::node()/child::book}}_{\text{descendant :: book}}$

18

XPath with //, ancestor, @, and text()

`//book//author[./text()="Knuth"]`

location step **3**: "descendant :: author"

```
SELECT r3.pre FROM book_tbl r1, r2, r3
           WHERE r1.pre=0
           AND r2.pre>r1.pre
           AND r2.post<r1.post
           AND r2.tag="book"
           AND r3.pre>r2.pre
           AND r3.post<r2.post
           AND r3.tag="author"
           AND
```

19

XPath with //, ancestor, @, and text()

`//book//author[./text()="Knuth"]`

plus filter

```
SELECT r3.pre FROM book_tbl r1, r2, r3, r4
           WHERE r1.pre=0
           AND r2.pre>r1.pre
           AND r2.post<r1.post
           AND r2.tag="book"
           AND r3.pre>r2.pre
           AND r3.post<r2.post
           AND r3.tag="author"
           AND r4.pre=r3.pre+1
           AND r4.post<r3.post
           AND r4.text="Knuth"
```

Cave
Not fully correct!!
Works only if author-nodes have
a single text node, as first child…

20

XPath with //, ancestor, @, and text()

`//book//author[./text()="Knuth"]`

plus filter

```
SELECT r3.pre FROM book_tbl r1, r2, r3, r4
           WHERE r1.pre=0
           AND r2.pre>r1.pre
           AND r2.post<r1.post
           AND r2.tag="book"
           AND r3.pre>r2.pre
           AND r3.post<r2.post
           AND r3.tag="author"
           AND r4.pre>r3.pre
           AND r4.post<r3.post
           AND r4.level=r3.level+1
           AND r4.text="Knuth"
ORDERED BY r3.pre
```

Correct: returns result nodes in document order.

21

XPath with //, ancestor, @, and text()

`//book//author[@name="Knuth"]`

plus filter

```
SELECT r3.pre FROM book_tbl r1, r2, r3,
                   book_attr a
           WHERE r1.pre=0
           AND r2.pre>r1.pre
           AND r2.post<r1.post
           AND r2.tag="book"
           AND r3.pre>r2.pre
           AND r3.post<r2.post
           AND r3.tag="author"
           AND r3.pre=a.pre
           AND a.attr="name"
           AND a.value="Knuth"
ORDERED BY r3.pre
```

Correct: returns result nodes in document order. What about duplicates?

22

XPath with //, ancestor, @, and text()

`//book//author[@name="Knuth"]`

DISTINCT

plus filter

```
SELECT  r3.pre FROM book_tbl r1, r2, r3,
                   book_attr a
           WHERE r1.pre=0
           AND r2.pre>r1.pre
           AND r2.post<r1.post
           AND r2.tag="book"
           AND r3.pre>r2.pre
           AND r3.post<r2.post
           AND r3.tag="author"
           AND r3.pre=a.pre
           AND a.attr="name"
           AND a.value="Knuth"
    ORDERED BY r3.pre
```

What about duplicates? E.g. `//author/ancestor::*`

always remove duplicates!

23

XPath with //, ancestor, @, and text()

`//book//author[@name="Knuth"]`

DISTINCT

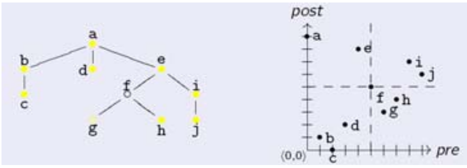
plus filter

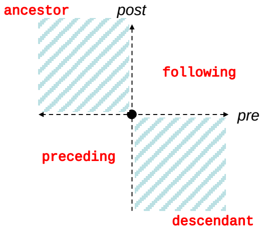
```
SELECT  r3.pre FROM book_tbl r1, r2, r3,
                   book_attr a
           WHERE r1.pre=0
           AND r2.pre>r1.pre
           AND r2.post<r1.post
           AND r2.tag="book"
           AND r3.pre>r2.pre
           AND r3.post<r2.post
           AND r3.tag="author"
           AND r3.pre=a.pre
           AND a.attr="name"
           AND a.value="Knuth"
    ORDERED BY r3.pre
```

→ Easy to add **following** and **preceding** axes.
What about other axes? E.g., **child, parent** etc??

24

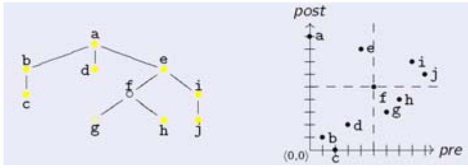
3. XPath with / and following-sibling

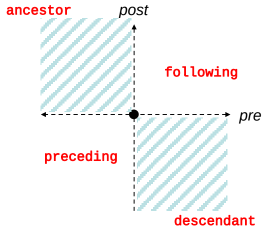




`firstChild( pr, po ) = ?`

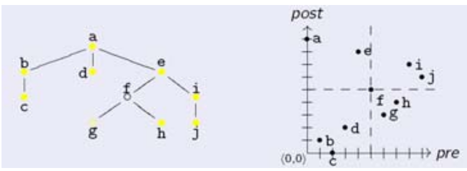
3. XPath with / and following-sibling

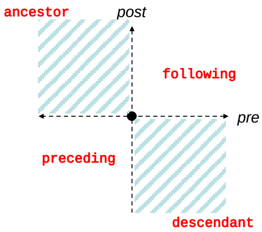




`firstChild( pr, po )` = *left-most node*,
below and to the right of (pr,po)

XPath with / and following-sibling



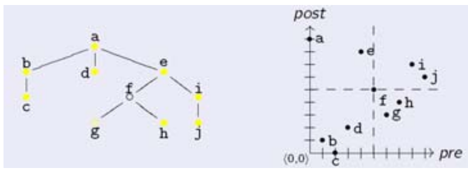


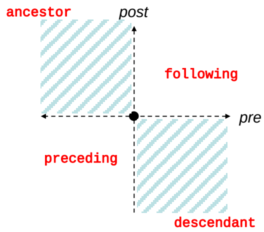
`firstChild( pr, po )` = *left-most node*,
below and to the right of (pr,po)

Not good!

Needs
`SELECT min(pre) FROM ..`

XPath with / and following-sibling



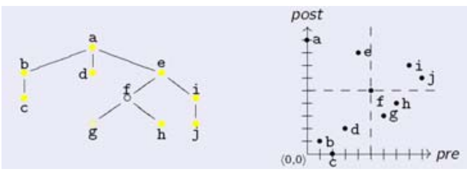


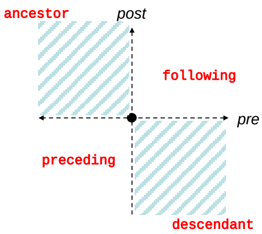
`firstChild( pr, po )` = *left-most node*,
below and to the right of (pr,po)

But easy: ☺

```
.. AND r2.pre=r1.pre+1
   AND r2.post<r1.post
```


XPath with / and following-sibling



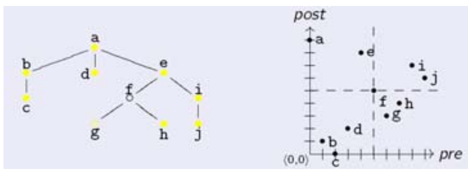


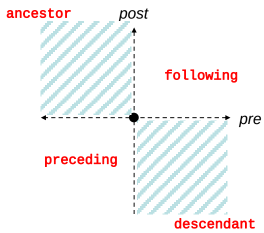
How to select **ALL children** of a node?

But easy

```
.. AND r2.pre=r1.pre+1
   AND r2.post<r1.post
```


XPath with / and following-sibling

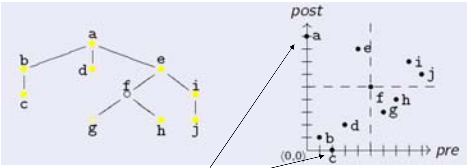


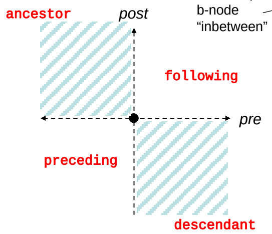


How to select **ALL children** of a node?

maybe
"descendant, i.e., larger pre and smaller post,
AND no other descendant inbetween"

XPath with / and following-sibling



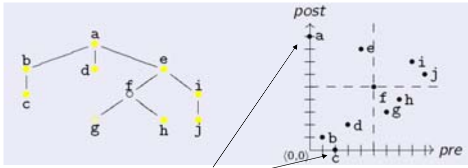


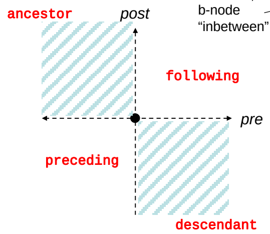
How to select **ALL children** of a node?

maybe
“descendant, i.e., larger pre and smaller post,
AND no other descendant inbetween”

can be done in SQL, but is **expensive**!
`… AND NOT EXIST (SELECT …`

XPath with / and following-sibling

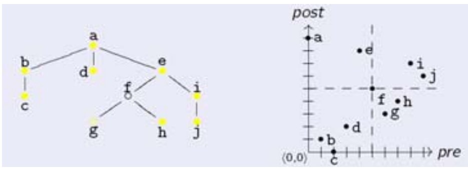


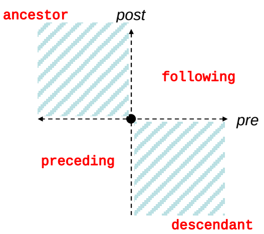


How to select **ALL children** of a node?

→ Using LEVEL this is easy:

Select all descendants at LEVEL +1 …

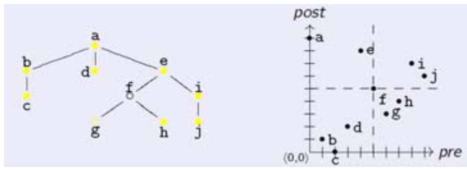


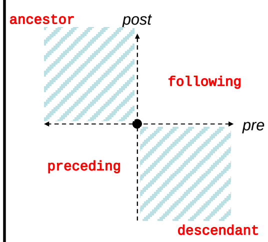


`firstChild`(pr, po) = left-most node,
below and to the right of (pr,po)

`nextSibling`(pr, po) =
left-most node (pr2, po2),
→ to the right
→ up
such that there is no node
with post value > po and < po2.

e.g., **not** c- and d-node
(because b-node is inbetween..)

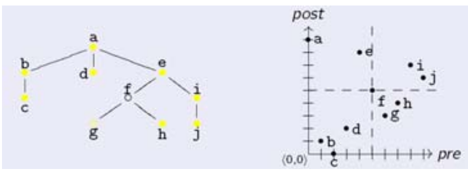


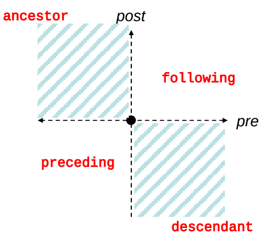


`firstChild`(pr, po) = left-most node,
below and to the right of (pr,po)

`following-sibling`

Can be done similar as before, using
a complicated SQL query.
→ Expensive





`firstChild`(pr, po) = left-most node,
below and to the right of (pr,po)

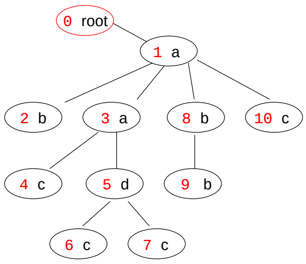
`following-sibling`

Can be doen similar as before, using
a complicated SQL query.
→ Expensive

→ Use other encoding!!

PRE / SIZE / LEVEL

Pre / Size / Level Encoding



pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

Question

Can you compute the post-value of a node,
from its (pre, size, level) values?

Pre / Size / Level Encoding

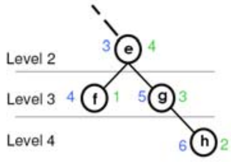
Useful relationships & approximations

Knowledge taken from specific properties of pre/post ranks to shrink window

size(**e**) = level(**e**) - pre(**e**) + post(**e**) (1)
3 = 2 - 3 + 4

For the right most leaf of the sub tree we can say

size(**e**) = pre(**h**) - pre(**e**) (2)
3 = 6 - 3



Pre / Size / Level Encoding

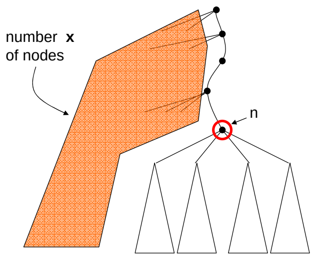
Useful relationships & approximations

THUS pre(n) + size(n) = post(n) + level(n)
Why?

Pre / Size / Level Encoding

Useful relationships & approximations

THUS pre(n) + size(n) = post(n) + level(n)
Why?



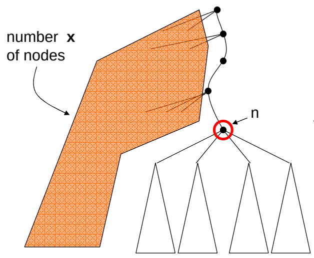
pre(n) = **x** + level(n)

post(n) = **x** + size(n)

Pre / Size / Level Encoding

Useful relationships & approximations

THUS pre(n) + size(n) = post(n) + level(n)
Why?



pre(n) = **x** + level(n)

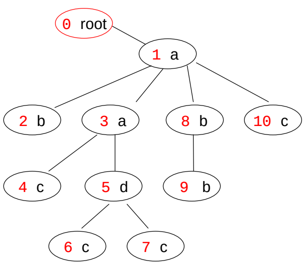
- post(n) = **x** + size(n)

pre(n) - post(n) = level(n) - size(n)

Pre / Size / Level Encoding

Useful relationships & approximations

THUS pre(n) + size(n) = post(n) + level(n)
Why?



pre	size	post	level	lab
0	10	10	0	root
1	9	9	1	a
2	0	0	2	b
3	4	5	2	a
4	0	1	3	c
5	2	4	3	d
6	0	2	4	c
7	0	3	4	c
8	1	7	2	b
9	0	6	3	b
10	0	8	2	c

Pre / Size / Level Encoding

Useful relationships & approximations

From equation (2) formed to
pre(**h**) = pre(**e**) + size(**e**) and replacing size(**e**) with (1)
leads to pre (**h**) = post(**e**) + level(**e**)

For a useful estimation we can also for sure say that:
Level(**e**) ≤ height(**t**)

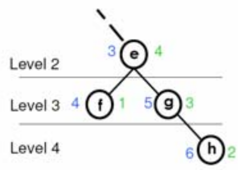
Using height(t) instead of level(e):

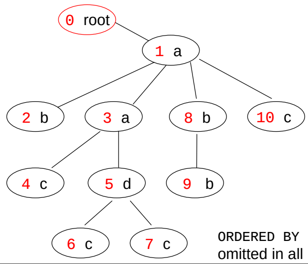
Node **h** has max pre-order and node **f** has min post order rank

pre(**h**) ≤ post(**e**) + height(**t**)

post(**f**) ≥ pre(**e**) – height(**t**)

The value height(t) of a document tree is assigned at document loading time.





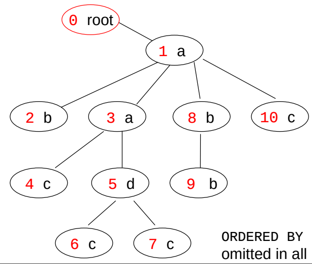
pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

ORDERED BY omitted in all queries

descendant(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE r1.pre >  pr
          AND r1.pre <= pr+si
```

43



pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

ORDERED BY omitted in all queries

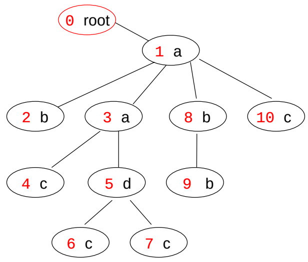
descendant(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE r1.pre >  pr
          AND r1.pre <= pr+si
```

ancestor(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
    WHERE r1.pre <  pr
      AND r1.pre+r1.size >= pr
```

44



pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

descendant(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE r1.pre >  pr
          AND r1.pre <= pr+si
```

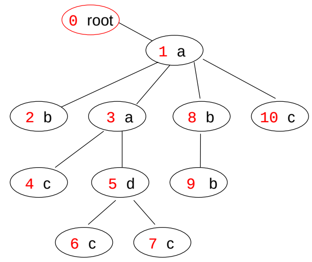
ancestor(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
    WHERE r1.pre <  pr
      AND r1.pre+r1.size >= pr
```

child(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE
```

45



pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

descendant(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE r1.pre >  pr
          AND r1.pre <= pr+si
```

“d”

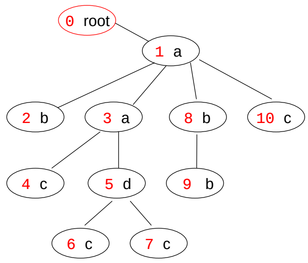
ancestor(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
    WHERE r1.pre <  pr
      AND r1.pre+r1.size >= pr
```

child(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE  “d”
          AND r1.level = le+1
```

46



pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

descendant(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE r1.pre >  pr
          AND r1.pre <= pr+si
```

“d”

ancestor(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
    WHERE r1.pre <  pr
      AND r1.pre+r1.size >= pr
```

“a”

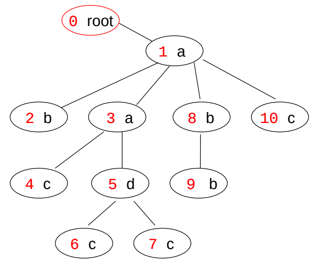
child(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE  “d”
          AND r1.level = le+1
```

parent(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE  “a”
          AND r1.level = le-1
```

47

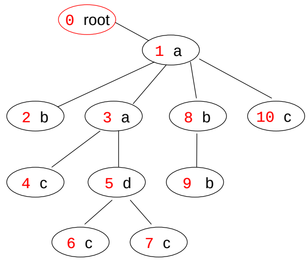


pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

preceding(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
    WHERE r1.pre+r1.size < pr
```

48



pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

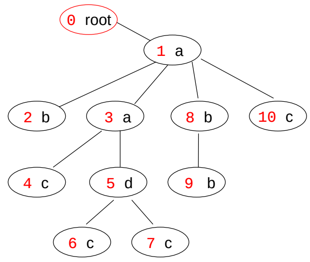
preceding(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
      WHERE r1.pre+r1.size < pr
```

following(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
    WHERE r1.pre > pr+si
```

49



pre	size	level	lab
0	10	0	root
1	9	1	a
2	0	2	b
3	4	2	a
4	0	3	c
5	2	3	d
6	0	4	c
7	0	4	c
8	0	2	b
9	0	3	b
10	0	2	c

preceding(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
      WHERE r1.pre+r1.size < pr
```


following-sibling(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
        WHERE  ??
```


following(pr, si, le) =

```
SELECT r1.pre FROM doc_tbl r1
    WHERE r1.pre > pr+si
```


50

Sometimes even store Parent with Pre / Size / Level

XPath Axis	Axis Conditions
self	$pre(x) = pre(y) \wedge kind(y) \neq att$
attribute	$parent(y) = pre(x) \wedge kind(y) = att$
parent	$parent(x) = pre(y) \wedge kind(y) \neq att$
child	$parent(y) = pre(x) \wedge kind(y) \neq att$
descendant	$pre(y) > pre(x) \wedge pre(y) \leq pre(x) + size(x) \wedge kind(y) \neq att$
descendant-or-self	$pre(y) \geq pre(x) \wedge pre(y) \leq pre(x) + size(x) \wedge kind(y) \neq att$
ancestor	$pre(y) < pre(x) \wedge pre(x) \leq pre(y) + size(y) \wedge kind(y) \neq att$
ancestor-or-self	$pre(y) \leq pre(x) \wedge pre(x) \leq pre(y) + size(y) \wedge kind(y) \neq att$
following	$pre(y) > pre(x) + size(x) \wedge kind(y) \neq att$
following-sibling	$pre(y) > pre(x) \wedge parent(y) = parent(x) \wedge kind(y) \neq att$
preceding	$pre(y) + size(y) < pre(x) \wedge kind(y) \neq att$
preceding-sibling	$pre(y) < pre(x) \wedge parent(y) = parent(x) \wedge kind(y) \neq att$

Level not needed, if we have parent.

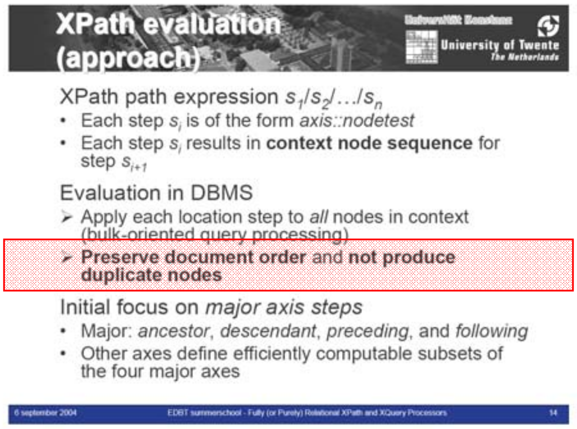
51

4. Staircase Join

Most of the following slides are by
Maurice van Keulen (Univ. of Twente, The Netherlands)

See
http://edbtss04.dia.uniroma3.it/VanKeulen.pdf

52



- -

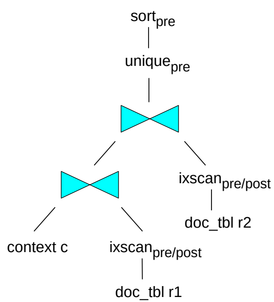
4. Staircase Join

Context/following::*/descendant::*

SQL Query

```
SELECT DISTINCT r2.pre
       FROM context c,
            doc_tbl r1, r2
      WHERE r1.pre   > c.pre
        AND r1.post  > c.post
        AND r2.pre   > r1.pre
        AND r2.post  < r1.post
ORDERED BY r2.pr
```

IBM DB2 Query Plan



54

4. Staircase Join

Context/following::*/descendant::*

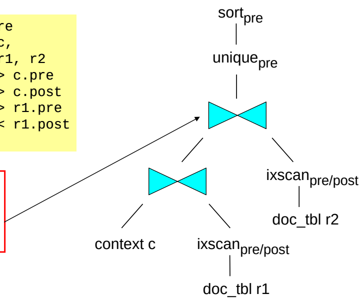
SQL Query

```
SELECT DISTINCT r2.pre
       FROM context c,
            doc_tbl r1, r2
       WHERE r1.pre  > c.pre
         AND r1.post > c.post
         AND r2.pre  > r1.pre
         AND r2.post < r1.post
ORDERED BY r2.pr
```

Problem 1

might have **MANY** duplicates here!

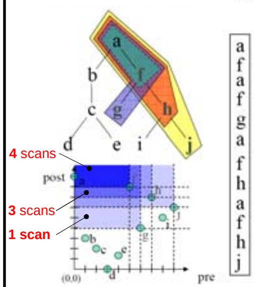
IBM DB2 Query Plan



4. Staircase Join

Technique 1 Avoiding Duplicates

(f,g,h,j)/ancestor-or-self::*



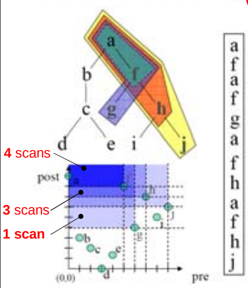
4. Staircase Join

Technique 1 Avoiding Duplicates

(f,g,h,j)/ancestor-or-self::*

Select *lowest independent* nodes

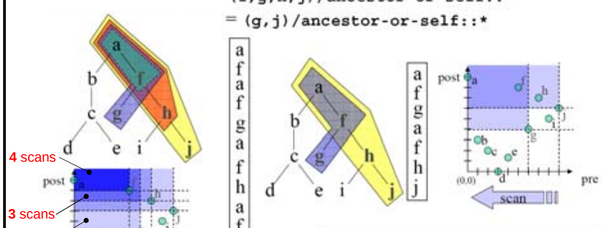
(not ancestor of each other)



4. Staircase Join

Technique 1 Pruning

(f,g,h,j)/ancestor-or-self::*
= (g,j)/ancestor-or-self::*



Pruning requires only a single sequential scan over the *pre/post* table.

4. Staircase Join

Technique 1 Pruning

→ How to Prune depends on axis

C set of context nodes
ax XPath axis

(1) If ax=ancestor, then
Prune(C,ax) = *lowest (=bottom-most) independent* nodes in C

(2) If ax=descendant, then

4. Staircase Join

Technique 1 Pruning

→ How to Prune depends on axis

C set of context nodes
ax XPath axis

(1) If ax=ancestor, then
Prune(C,ax) = *lowest (=bottom-most) independent* nodes in C

(2) If ax=descendant, then
Prune(C,ax) = *highest (=top-most) independent* nodes in C

4. Staircase Join

Technique 1 Pruning

→ How to Prune depends on axis

C set of context nodes
ax XPath axis

(1) If ax=ancestor, then
Prune(C,ax) = *lowest independent* nodes in C

(2) If ax=descendant, then
Prune(C,ax) = *highest independent* nodes in C

Qu: give pruning rule for

(3) ax = following

?

4. Staircase Join

Technique 1 Pruning

Hint For any two context nodes N1 and N2:

following-nodes(N1) ⊆ following-nodes(N2)
OR following-nodes(N2) ⊆ following-nodes(N1)

Why is that?
Consider arbitrary N1, N2.
Either one is a descendant of the other or…

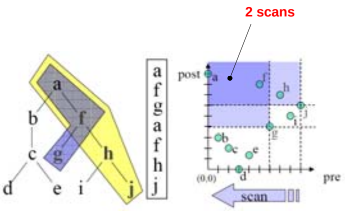
Qu: give pruning rule for

(3) ax = following

?

4. Staircase Join

Technique 2 Partitioning

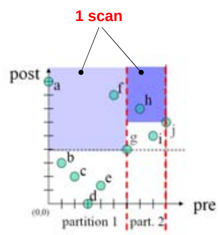


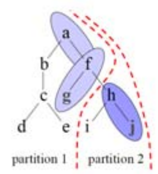
Problem

Still 2-scan area that generates duplicates

4. Staircase Join

Technique 2 Partitioning





4. Staircase Join

(b,c,h) $\bowtie_{desc}$ *doc*

```
context ⋈desc doc =
begin
  result←new table(pre,post);
  /* partition c_from ... c_to */
  c_from ← first node in context;
  while (c_to ← next node in context) do
    if c_to.post < c_from.post then
      /* prune */
    else
      scanpartition_desc(c_from.pre+1,c_to.pre-1,
        c_from.post);
      c_from ← c_to;
  n←last node in doc;
  scanpartition_desc(c_from.pre+1,n.pre,c_from.post);
  return result;
end

scanpartition_desc(pre1,pre2,post)
begin
  for i from pre1 to pre2 do
    if doc[i].post < post then
      append doc[i] to result;
end
```


4. Staircase Join

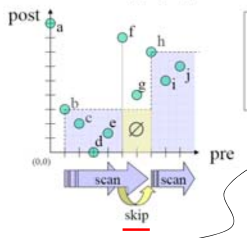
1) It scans doc and context tables *sequentially*
2) It scans both tables only *once* for an entire context node sequence
3) It *never* delivers duplicate nodes
4) Result nodes are produced in *document order*
5) Input for staircase join can be *any* node sequence

➢ (3) + (4)
⇒ no post-processing (unique/sort) is needed to comply to XPath semantics

4. Staircase Join

Technique 2 Partitioning & Skipping

`(b,c,h)/descendant::*`




```
scanpartition_desc(pre1,pre2,post)
begin for i from pre1 to pre2 do
        if doc[i].post < post then
           append doc[i] to result;
        else break;  /* skip */
end
```

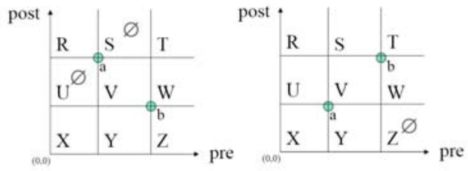
- Never touch more than |*context*| + |*result*| nodes
- A characterization of the location of 'f' can be found **before** scan starts (*estimated* skipping)

CAVE: only true if there are no value comparisons in filters…

4. Staircase Join

More.. Skip *empty regions* in the plane

For *any* two nodes 'a' and 'b'



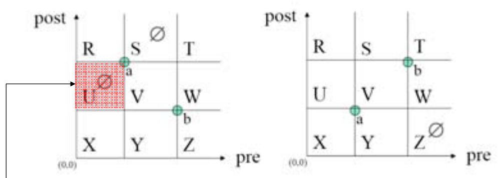
(1) Nodes *a* and *b* relate to each other on the ancestor/descendant axis.

(2) Nodes *a* and *b* relate to each other on the preceding/following axis.

4. Staircase Join

More.. Skip *empty regions* in the plane

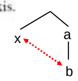
For *any* two nodes 'a' and 'b'



(1) Nodes *a* and *b* relate to each other on the ancestor/descendant axis.

(2) Nodes *a* and *b* relate to each other on the preceding/following axis.

Why? → *b* cannot be a descendant of a preceding node of *a*!

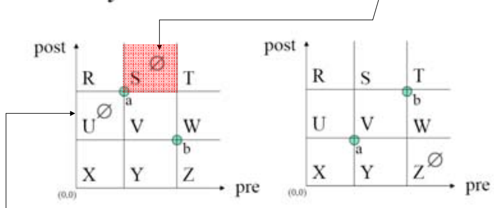


4. Staircase Join

Why?

More.. Skip *empty regions* in the plane

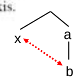
For *any* two nodes 'a' and 'b'



(1) Nodes *a* and *b* relate to each other on the ancestor/descendant axis.

(2) Nodes *a* and *b* relate to each other on the preceding/following axis.

Why? → *b* cannot be a descendant of a preceding node of *a*!



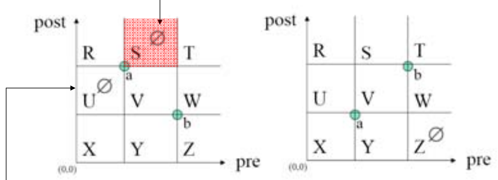
4. Staircase Join

Why?

→ a's following nodes are also following nodes of b.

More.. Skip *empty regions* in the plane

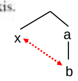
For *any* two nodes 'a' and 'b'



(1) Nodes *a* and *b* relate to each other on the ancestor/descendant axis.

(2) Nodes *a* and *b* relate to each other on the preceding/following axis.

Why? → *b* cannot be a descendant of a preceding node of *a*!



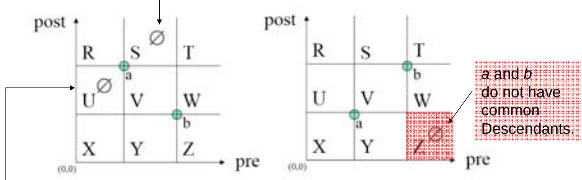
4. Staircase Join

Why?

→ a's following nodes are also following nodes of b.

More.. Skip *empty regions* in the plane

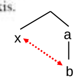
For *any* two nodes 'a' and 'b'



(1) Nodes *a* and *b* relate to each other on the ancestor/descendant axis.

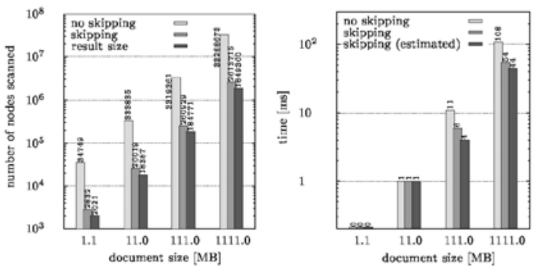
(2) Nodes *a* and *b* relate to each other on the preceding/following axis.

Why? → *b* cannot be a descendant of a preceding node of *a*!



Experiments: effect of skipping

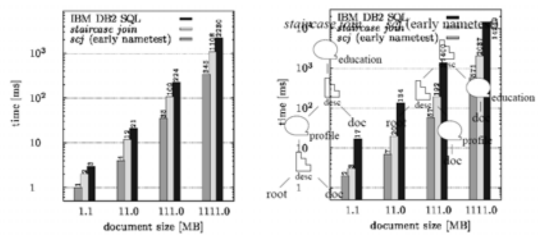
Universität Konstanz
University of Twente
The Netherlands



73

Experiments: Tree-unaware vs. Tree-aware

Universität Konstanz
University of Twente
The Netherlands



Query 1: `/descendant::profile/descendant::education`

Query 2: `/descendant::increase/ancestor::bidder`

74

END
Lecture 10

75