Storing XML on Relational Tables

Federico Ulliana
GraphIK, LIRMM, INRIA

Slides collected from James Cheney and Sam Idicula

Boston, winter ’99 : XML standardization

Jan 2000 : people wondering …

Now, how can I publish online my relational data?
(XMLAGG – Xperanto)

Feb 2000 : people (again) wondering …

I created my first10GB XML document crawling web data.
Now, how can I query it ?

3 schools for processing XML data

1. Flat streams: store XML data as is in text files
   - query support: limited; fast for retrieving whole documents

2. Native XML Databases: designed specifically for XML
   - XML document stored in XML specific way
   - Goal: Efficient support for XML queries

3. Re-use existing DB storage systems
   - Leverage mature systems (DBMS)
   - How ? Map XML document into flat tables

Why transform XML data into relations?

Native XML databases need:

- storing XML data, indexing,
- query processing/optimization
- concurrency control
- updates
- access control, . . .
- Nontrivial: the study of these issues is still in its infancy – incomplete support for general data management tasks

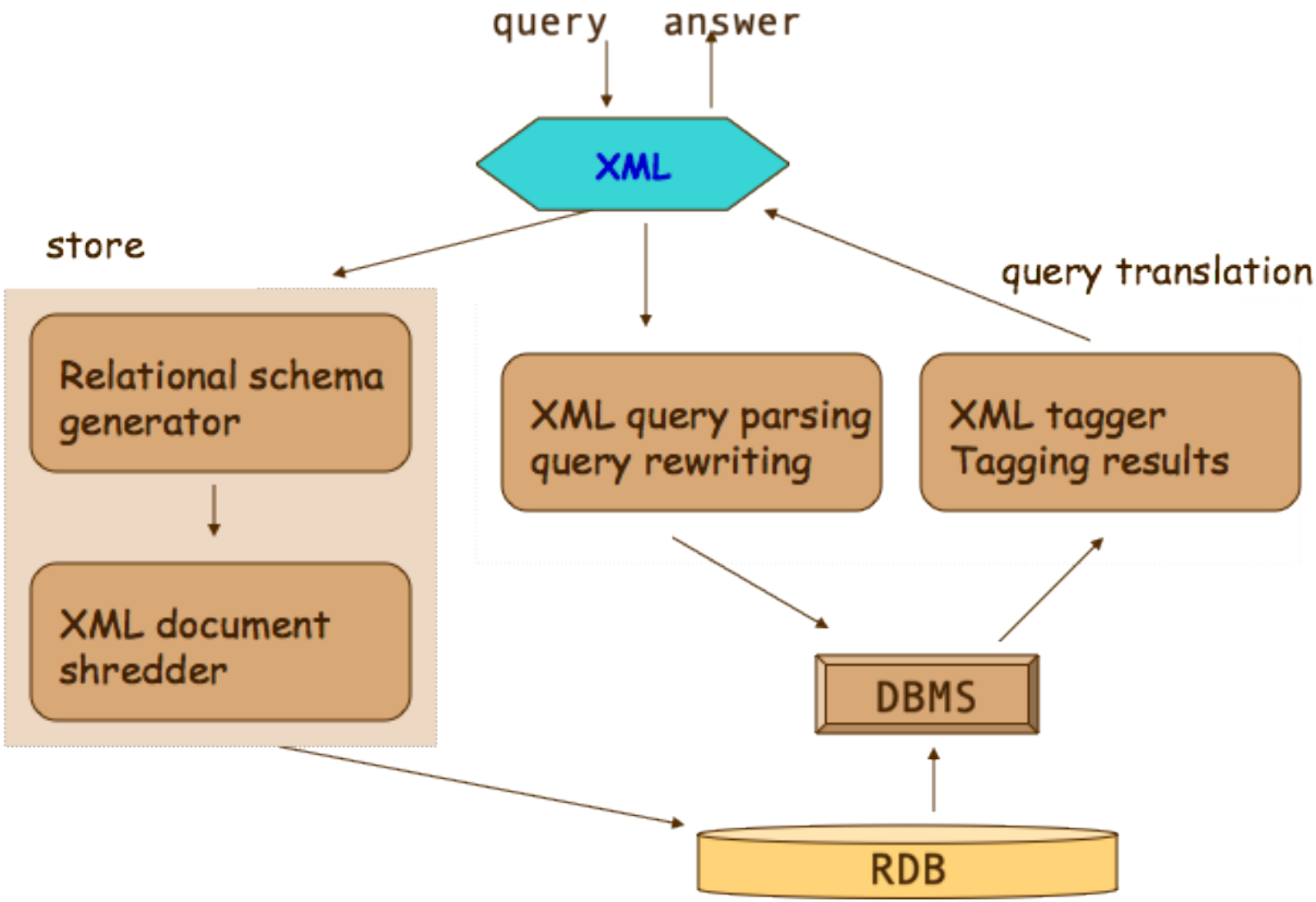
Haven't these already been developed for relational DBMS!?

- Why not take advantage of available DBMS techniques?

From XML to relations :

1. Derive a relational schema
2. Insert XML data into relational tuples
3. Translate XML queries to SQL queries
4. Convert query results back to XML

Architecture



Nontrivial issues

Data model mismatch

- DTD: recursive, regular expressions/nested content
- relational schema: tables, single-valued attributes

Information preservation

- lossless: there should be an effective method to reconstruct the original XML document from its relational storage
- propagation/preservation of integrity constraints

Query language mismatch

- XQuery, XSLT: Turing-complete
- XPath: transitive edges (descendant, ancestor)
- SQL: first-order, limited / no recursion

Plan

Schema-unaware

Schema-aware

Commercial solutions

SCHEMA-UNAWARE XML STORAGE

Schema-unaware storage

Storage easier if we have a fixed schema

But, often don't have schema

Or schema may change over time

- schema updates require reorganizing or reloading!

So: schema-oblivious XML storage

Schema chaos: In this scenario, customers want the flexibility to manage XML data that may or may not have schema, or may have "any" schema. For instance, a telecommunication customer wants to manage XML data generated from different towers, which generate documents with slightly different schemas from each other. They want to store them in one table and perform efficient query on the shared common pieces.

Schema-unaware storage

- Edge
- Vertical-Edge
- Monet
- Intervals

The basics first

"before thinking about sophisticated solutions, how the simplest and most obvious approaches perform?"

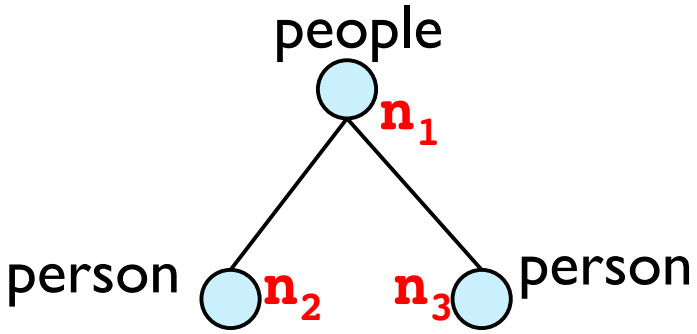
Round 1) EDGE vs VERTICAL-EDGE

EDGE storage

Observation: XML ordered trees can be encoded with

binary relation `EDGE(parent,child)`

order relation `NEXT-SIBLING(prec,succ)`

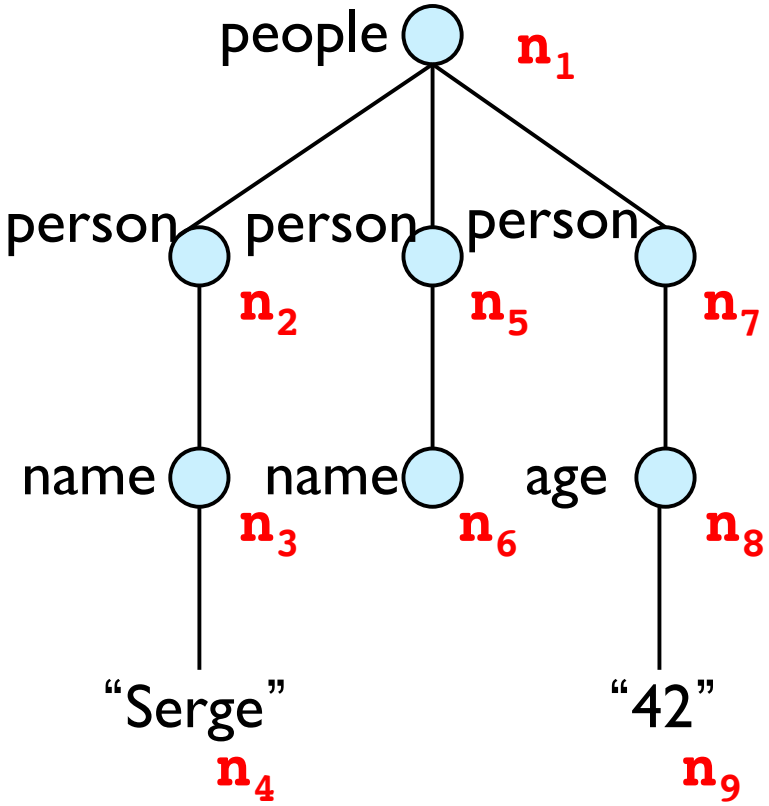


$\texttt{EDGE}(n_1, n_2)$

$\texttt{EDGE}(n_1, n_3)$

$\texttt{NEXT-SIBLING}(n_2, n_3)$

Edges & Values



EDGES

source	target	ordinal	tag	type
	n_1		people	elt
n_1	n_2	1	person	elt
n_1	n_5	2	person	elt
n_1	n_7	3	person	elt
n_2	n_3	1	name	elt
n_3	n_4	1		txt
n_5	n_6	1	name	elt
n_7	n_8	1	age	elt
n_8	n_9	1		num

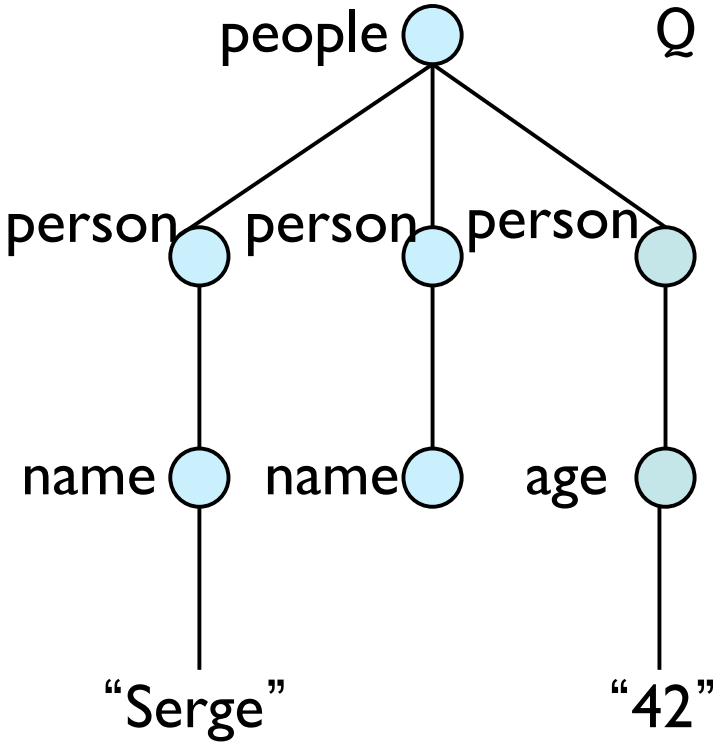
TEXTVALUES

node	value
n_4	Serge

NUMVALUES

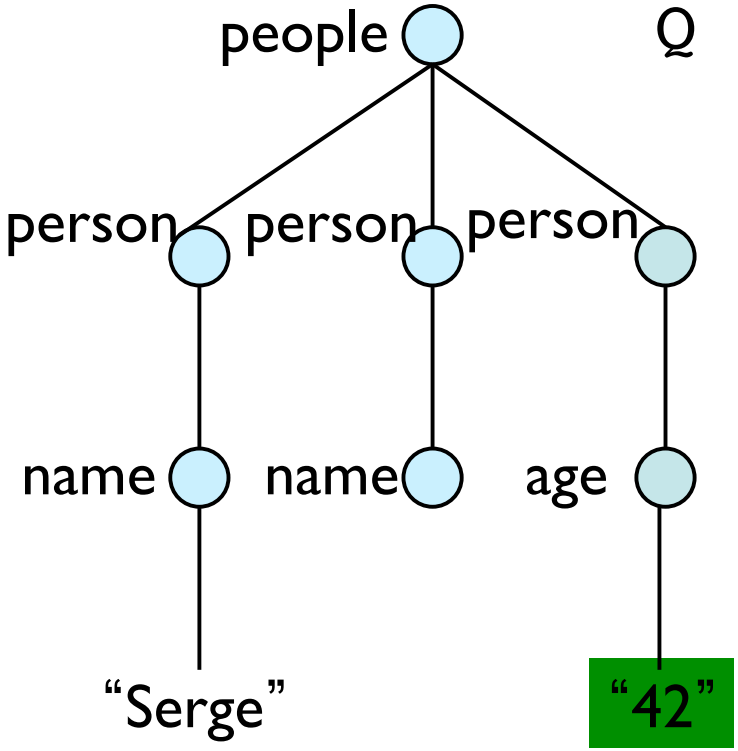
node	value
n_9	42

Querying



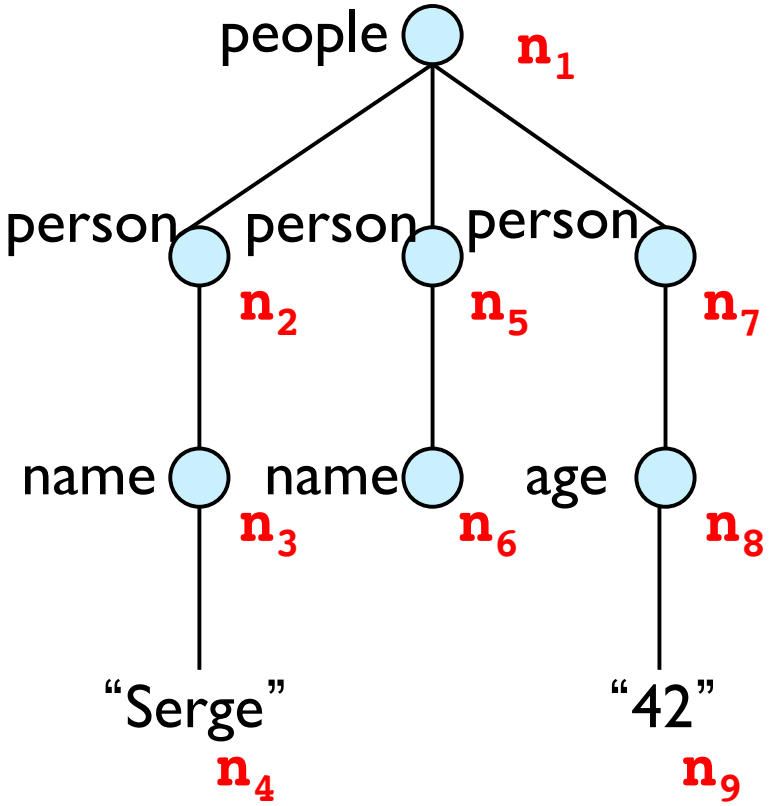

```
Q = /people/person/age/text()
```

Querying




```
Q = /people/person/age/text()
```

/people/person/age/text()



EDGES

source	target	ordinal	tag	type
	n_1		people	elt
n_1	n_2	1	person	elt
n_1	n_5	2	person	elt
n_1	n_7	3	person	elt
n_2	n_3	1	name	elt
n_3	n_4	1		txt
n_5	n_6	1	name	elt
n_7	n_8	1	age	elt
n_8	n_9	1		num

TEXTVALUES

node	value
n_4	Serge

NUMVALUES

node	value
n_9	42

/people/person/age/text()

```
SELECT N.value
FROM     EDGES e1,
         EDGES e2,
         EDGES e3,
         EDGES e4,
         NUMVALUES N
WHERE
         e1.target=e2.source
AND      e2.target=e3.source
AND      e3.target=e4.source
AND      e1.tag="people"
AND      e2.tag="person"
AND      e3.tag="age"
AND      e4.type="num"
AND      e4.target= N.node
```

EDGES

source	target	ordinal	tag	type
	n_1		people	elt
n_1	n_2	1	person	elt
n_1	n_5	2	person	elt
n_1	n_7	3	person	elt
n_2	n_3	1	name	elt
n_3	n_4	1		txt
n_5	n_6	1	name	elt
n_7	n_8	1	age	elt
n_8	n_9	1		num

TEXTVALUES

node	value
n_4	Serge

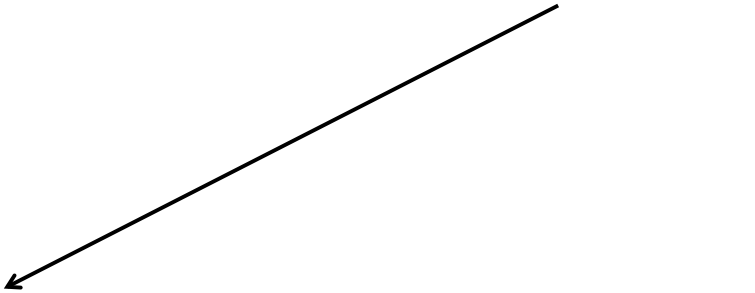
NUMVALUES

node	value
n_9	42

/people/person/age/text()

```
SELECT N.value
FROM     EDGES e1,
         EDGES e2,
         EDGES e3,
         EDGES e4,
         NUMVALUES N
WHERE
         e1.target=e2.source
AND      e2.target=e3.source
AND      e3.target=e4.source
AND      e1.tag="people"
AND      e2.tag="person"
AND      e3.tag="age"
AND      e4.type="num"
AND      e4.target= N.node
```

Lots of joins



TEXTVALUES

node	value
n_4	Serge

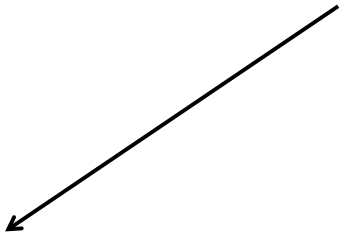
NUMVALUES

node	value
n_9	42

/people/person/age/text()

```
SELECT N.value
FROM     EDGES e1,
         EDGES e2,
         EDGES e3,
         EDGES e4,
         NUMVALUES N
WHERE
         e1.target=e2.source
AND      e2.target=e3.source
AND      e3.target=e4.source
AND      e1.tag="people"
AND      e2.tag="person"
AND      e3.tag="age"
AND      e4.type="num"
AND      e4.target= N.node
```

We also need a query testing for text values (UNION)



TEXTVALUES

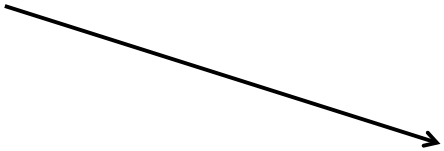
node	value
n_4	Serge

NUMVALUES

node	value
n_9	42

Querying

Fragmentation: tree spread across the table



EDGES

source	target	ordinal	tag	type
	n_1		people	elt
n_1	n_2	1	person	elt
n_1	n_5	2	person	elt
n_1	n_7	3	person	elt
n_2	n_3	1	name	elt
n_3	n_4	1		txt
n_5	n_6	1	name	elt
n_7	n_8	1	age	elt
n_8	n_9	1		num

TEXTVALUES

node	value
n_4	Serge

NUMVALUES

node	value
n_9	42

Querying

Fragmentation: tree spread across the table

Indexes **unaware** of tree structure

EDGES

source	target	ordinal	tag	type
	n_1		people	elt
n_1	n_2	1	person	elt
n_1	n_5	2	person	elt
n_1	n_7	3	person	elt
n_2	n_3	1	name	elt
n_3	n_4	1		txt
n_5	n_6	1	name	elt
n_7	n_8	1	age	elt
n_8	n_9	1		num

TEXTVALUES

node	value
n_4	Serge

NUMVALUES

node	value
n_9	42

How to improve ?

1. **Partitioning by row**
   group edges targeting same tag-label

2. **Inlining**
   put text and numeric values in the main table

EDGES

source	target	ordinal	tag	type
	n_1		people	elt
n_1	n_2	1	person	elt
n_1	n_5	2	person	elt
n_1	n_7	3	person	elt
n_2	n_3	1	name	elt
n_3	n_4	1		txt
n_5	n_6	1	name	elt
n_7	n_8	1	age	elt
n_8	n_9	1		num

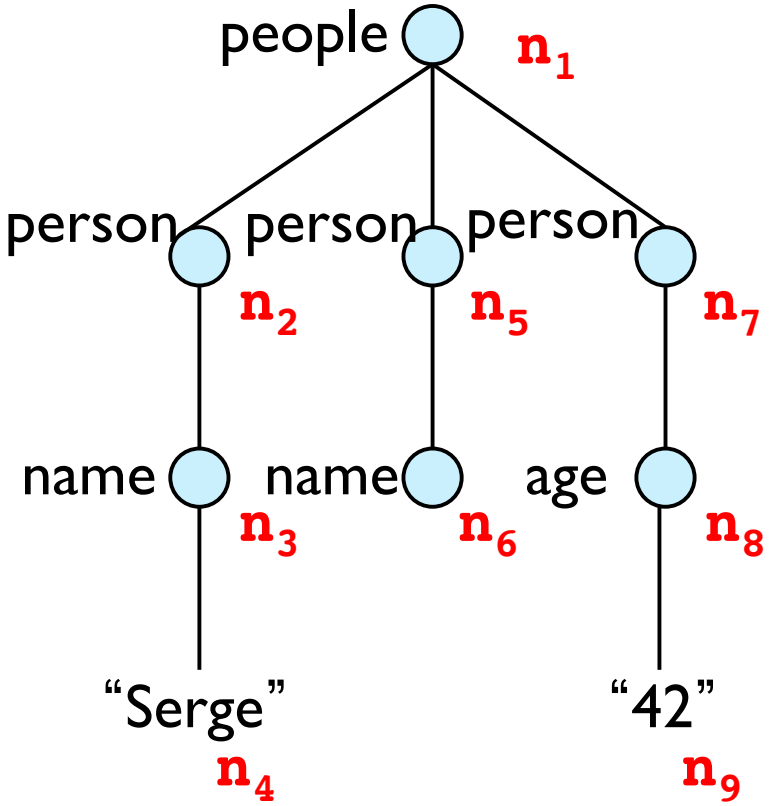
TEXTVALUES

node	value
n_4	Serge

NUMVALUES

node	value
n_9	42

VERTICAL-EDGE + Inline



people

source	target	ordinal	txtval	numval
	n_1			

person

source	target	ordinal	txtval	numval
n_1	n_2	1		
n_1	n_5	2		
n_1	n_7	3		

name

source	target	ordinal	txtval	numval
n_2	n_3	1	Serge	
n_5	n_6	1		

age

source	target	ordinal	txtval	numval
n_7	n_8	1		42

VERTICAL-EDGE + Inline

```
Q = /people/person/age/text()

SELECT  AGE.value
FROM     PEOPLE  P1,
         PERSON  P2,
         AGE

WHERE
         P1.target=P2.source
AND      P2.target=AGE.source
```

Joins on smaller tables

people

source	target	ordinal	txtval	numval
	n_1			

person

source	target	ordinal	txtval	numval
n_1	n_2	1		
n_1	n_5	2		
n_1	n_7	3		

name

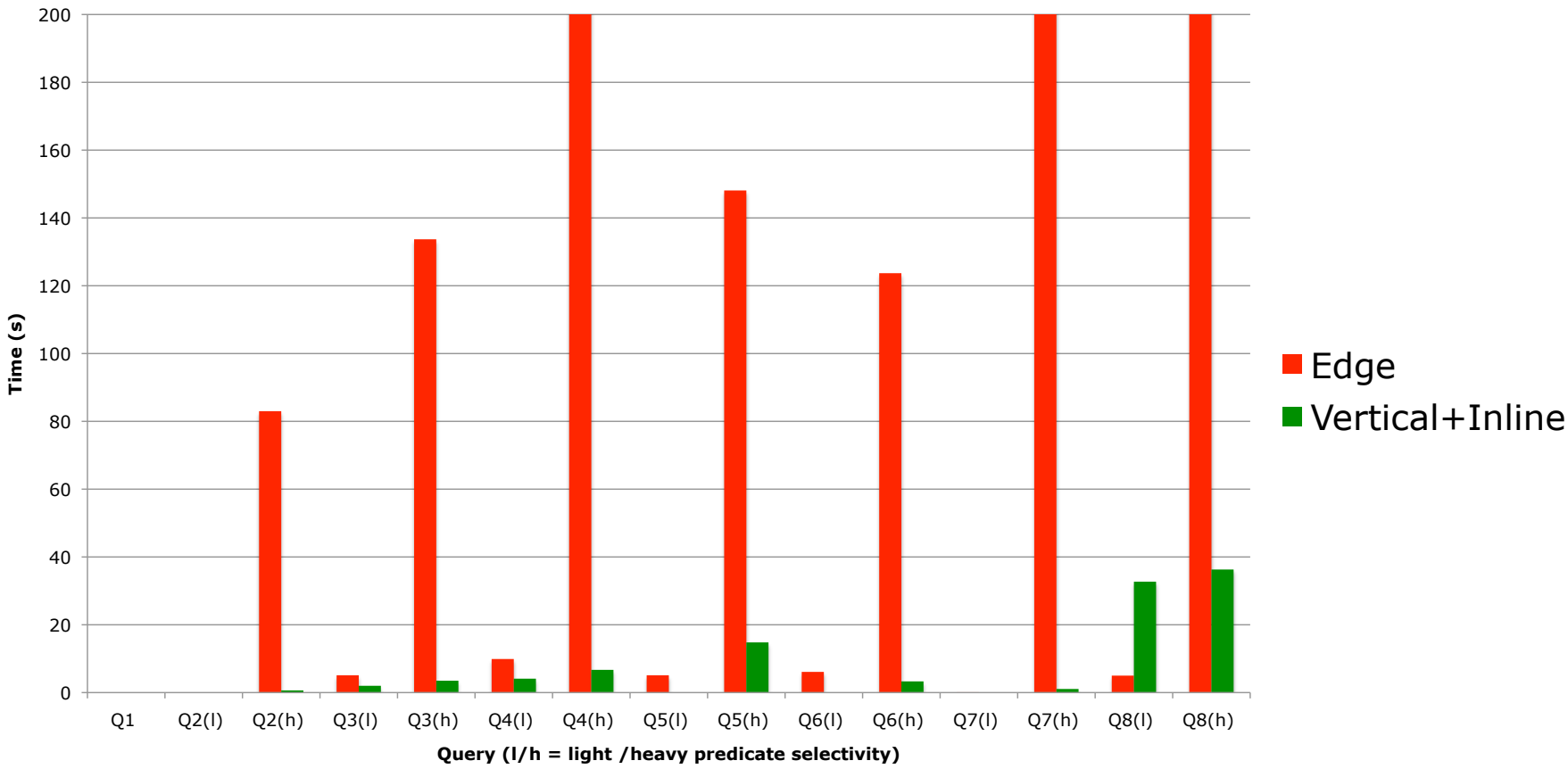
source	target	ordinal	txtval	numval
n_2	n_3	1	Serge	
n_5	n_6	1		

age

source	target	ordinal	txtval	numval
n_7	n_8	1		42

VERTICAL-EDGE+Inline beats EDGE

(query-answering time with the two storages)



The SQL queries you cannot ask

- Does it exists a direct flight between Paris and Los Angeles ? ✓
- Does it exists a (possibly indirect) flight between Montpellier and Austin ? ✗
  - problem : we do not know the number of intermediary airports (=joins)
- Does it exists a child for the node N ? ✓
- Is the node M a descendant of node N ? ✗
  - problem : we do not know the depth of a descendant node
  - taking max document depth and trying all possibilities is not an elegant solution

Issues with XPath axes

```
Q = /people//age/text()
```

Descendant = implicit recursion
sort of (child)*

Does not translate to
SELECT-FROM-WHERE query

Recursion :
ORACLE, POSTGRES OK
MySQL NO

people

source	target	ordinal	txtval	numval
	n_1			

person

source	target	ordinal	txtval	numval
n_1	n_2	1		
n_1	n_5	2		
n_1	n_7	3		

name

source	target	ordinal	txtval	numval
n_2	n_3	1	Serge	
n_5	n_6	1		

age

source	target	ordinal	txtval	numval
n_7	n_8	1		42

Limits of Edge/Vertical

Indexing unaware of tree structure

- fragmentation : subtree spread across db

Incomplete query translation

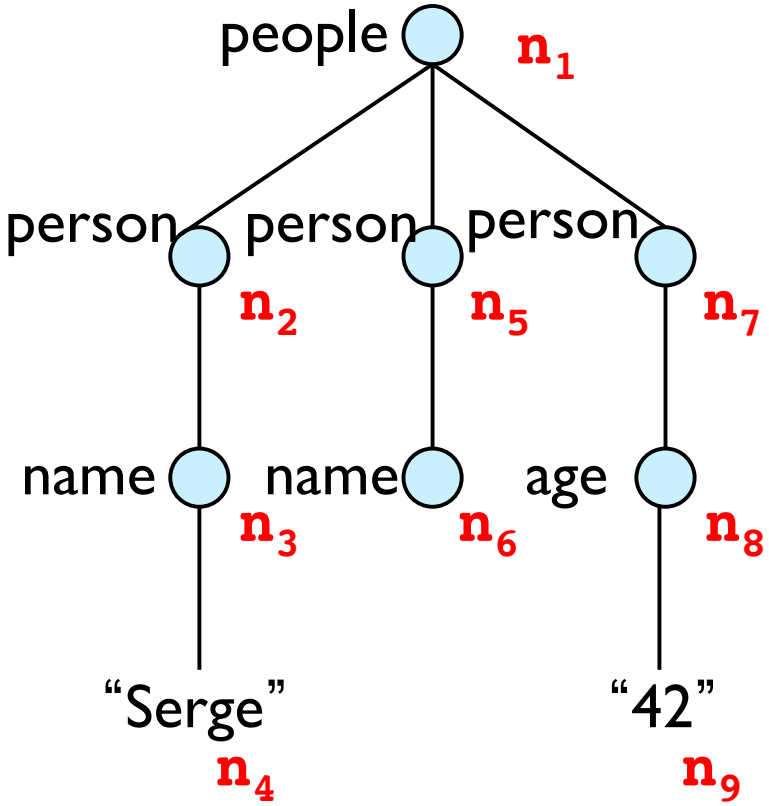
- descendant axis steps involve recursion

Lots of joins

- joins + no indexing = trouble

MONET storage

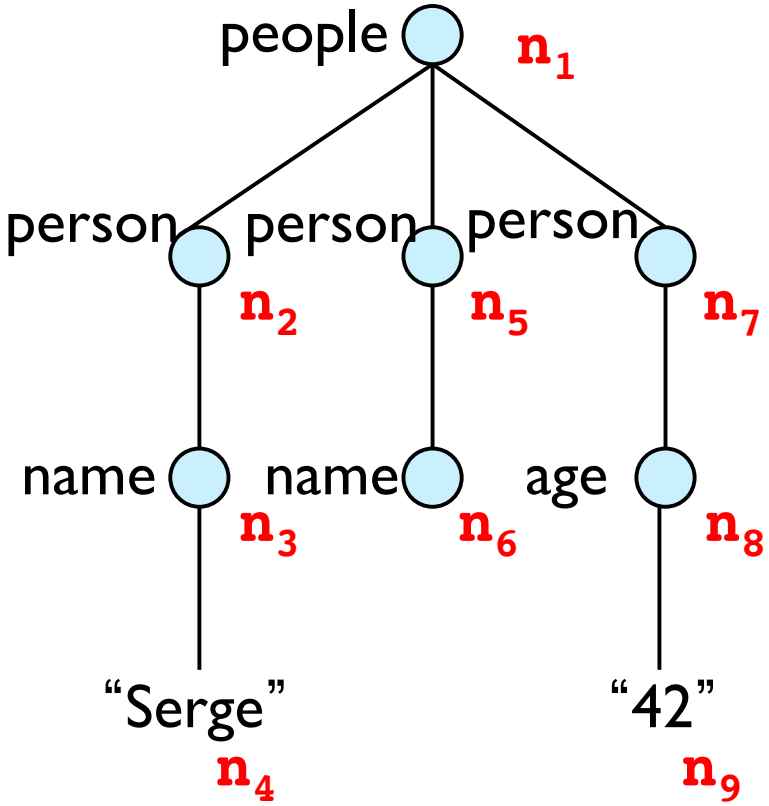
(so called because developed first on Monet-DB)



Idea : one table for each path in the XML tree

- `people`
- `people_person`
- `people_person_name`
- `people_person_age`

MONET storage



people

node	txtval	numval
n_1		

people_person

node	txtval	numval
n_2		
n_5		
n_7		

people_person_name

node	txtval	numval
n_3	Serge	
n_6		

people_person_age

node	txtval	numval
n_8		42

/people/person/age/text()

```
SELECT txtval,numval
FROM
people_person_age
```

people

node	txtval	numval
n_1		

people_person

node	txtval	numval
n_2		
n_5		
n_7		

people_person_name

node	txtval	numval
n_3	Serge	
n_6		

people_person_age

node	txtval	numval
n_8		42

Still one question…

And descendant axis ?

```
Q = /people//age
```

How to select the relations to query ?

/people//age

`people_(any-seq)_age`

people_*(any-seq)*_age

people	people_person
✗	✗
people_person_name	people_person_age
✗	✓

```
//person//*
```

(any-seq)`_person_`*(any-non-empty-seq)*

(any-seq)`_person_`*(any-non-empty-seq)*

*(any-seq)*_person_*(any-non-empty-seq)*

`people` ✗

`people_person` ✗

`people_person_name` ✓

`people_person_age` ✓

*(any-seq)*_person_*(any-non-empty-seq))*

```
SELECT node
FROM people_person_name
                                    UNION
SELECT node
FROM people_person_age
```

people_person_name people_person_age

✓ ✓

Performances

- MONET (obviously) beats VERTICAL-EDGE+Inlining

And the remaining axes ?