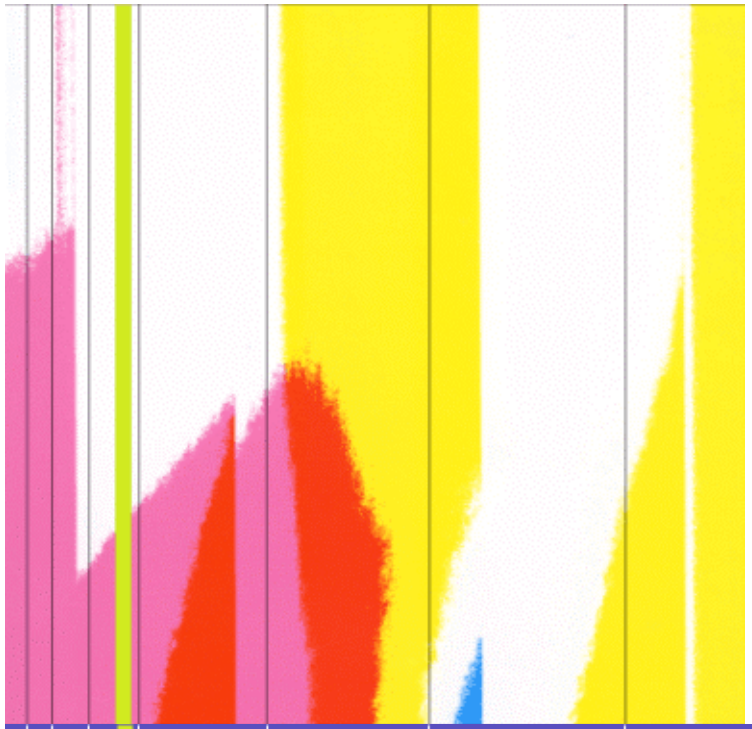


XML Access Modules: Towards Physical Data Independence for XML Databases

(ACI MD TRALALA)

Date

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

Andrei Arion — INRIA Futurs and Univ. Paris XI, France
Veronique Benzaken — Univ. Paris XI, France
Ioana Manolescu — INRIA Futurs, France
Ravi Vijay — IIT Bombay, India

Contexte: ACI TRALALA

TRAnsformations, Logiques et LAngages

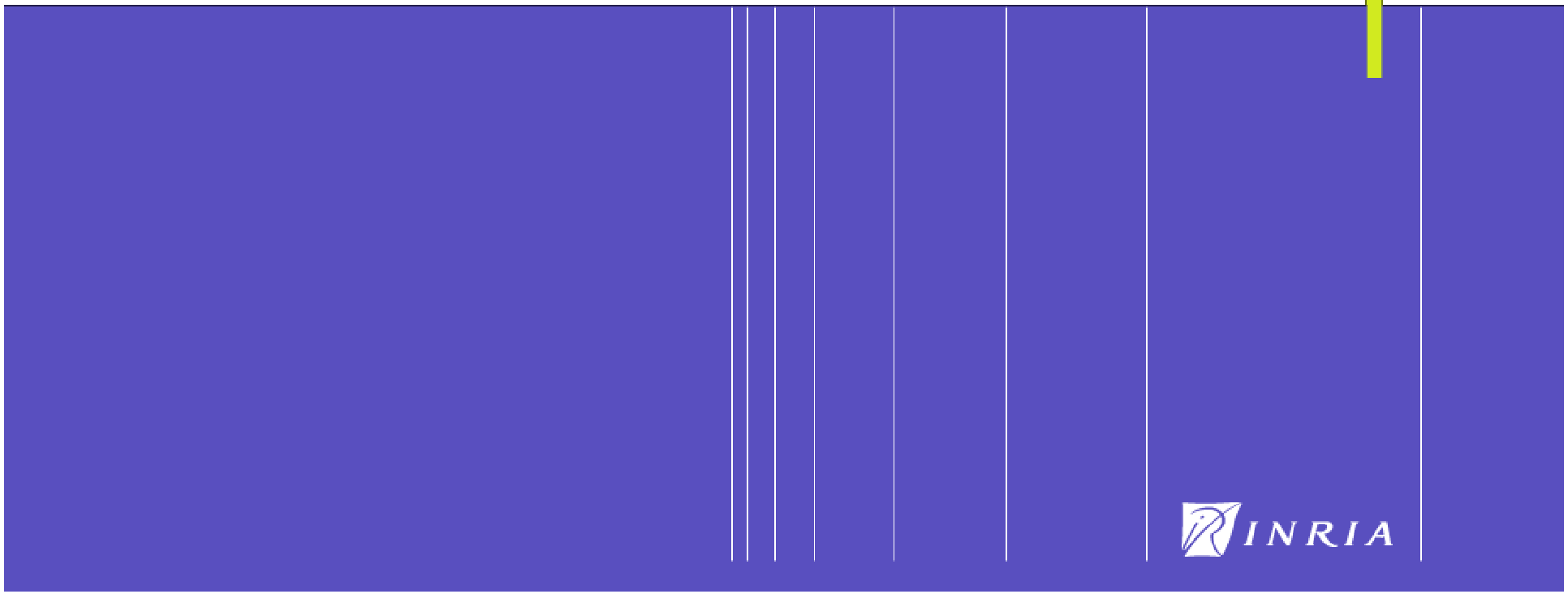
Theorie et pratique de l'interrogation de donnees XML:

- automates d'arbres
- langages fonctionnels, typage semantique (CQL et CDuce)
- complexite de (sous-ensembles de) XPath, XQuery
- compression XML
- traitements XML a la volee
- optimisation algebrique
- optimisation de chemin d'acces aux donnees

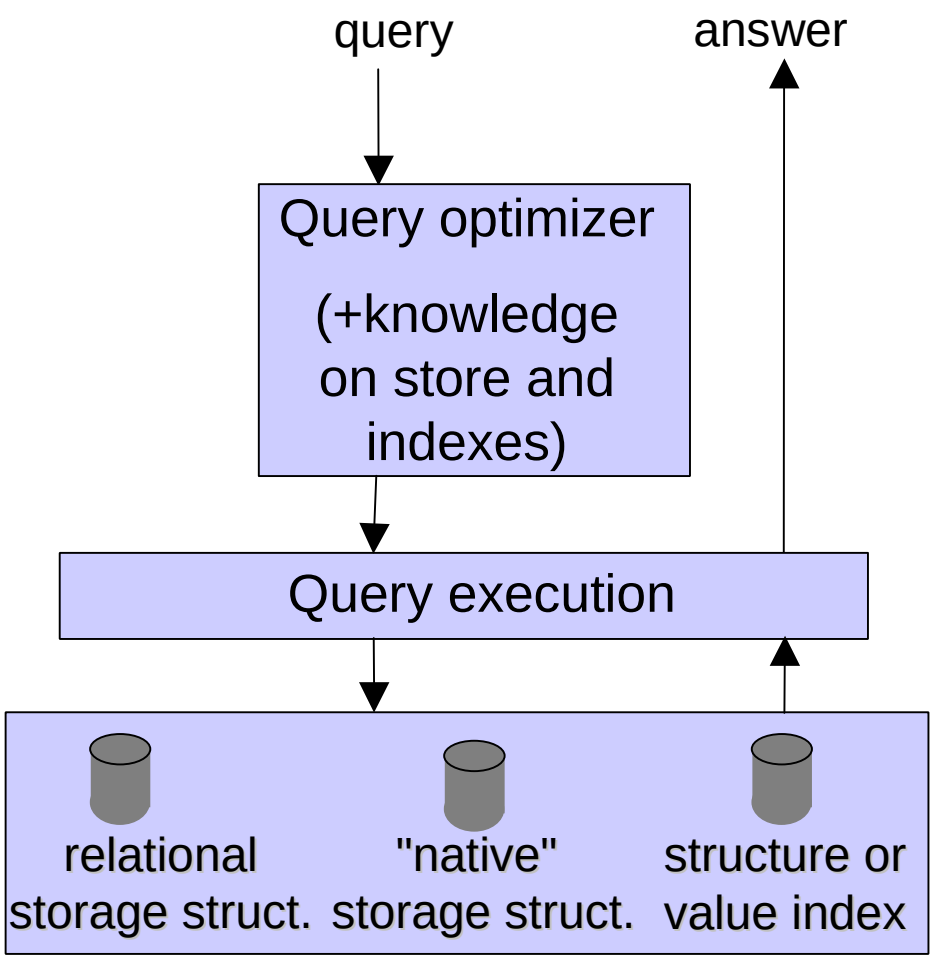
Plan

- The need for physical data independence in XML databases
- Our proposal: XML Access Modules (XAMs)
  - Algebraic language describing XML materialized views and indices
- Answering XQueries over XAMs
  - Constraint-based containment and rewriting
- Outline of a XAM-based XML DBMS architecture
- Conclusion

The need for physical data independence in XML databases

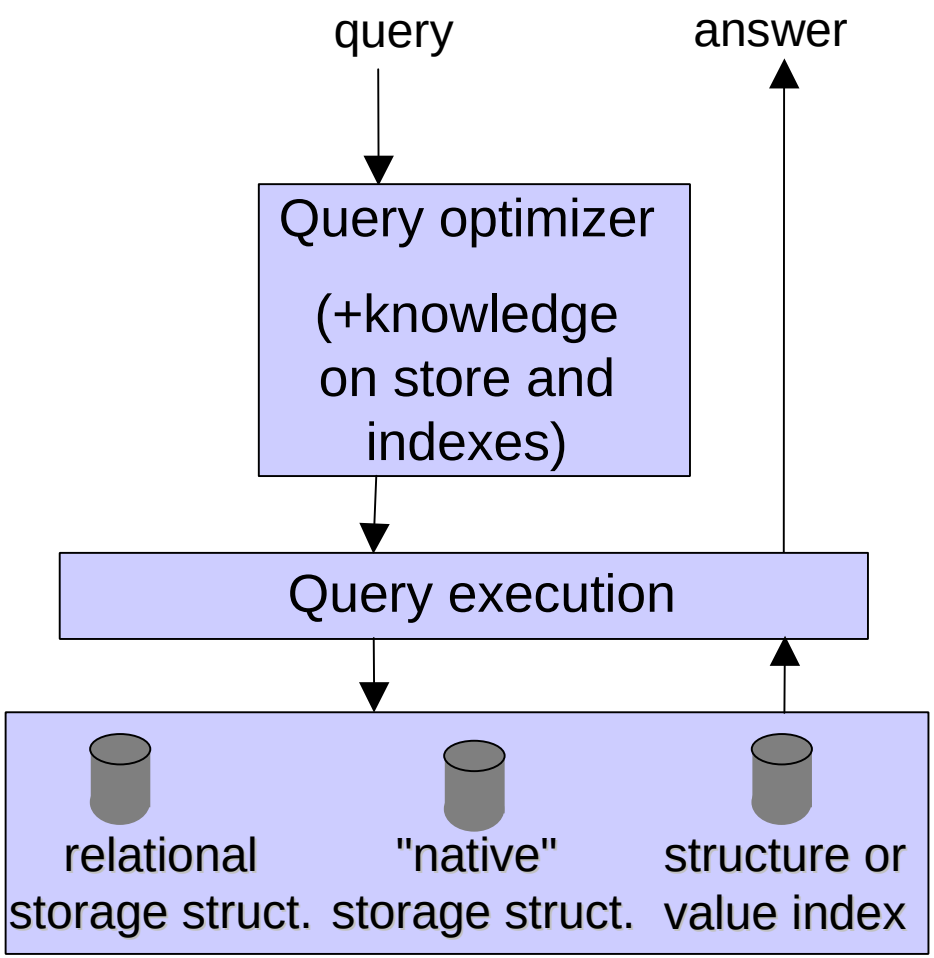


Query processing on XML persistent stores



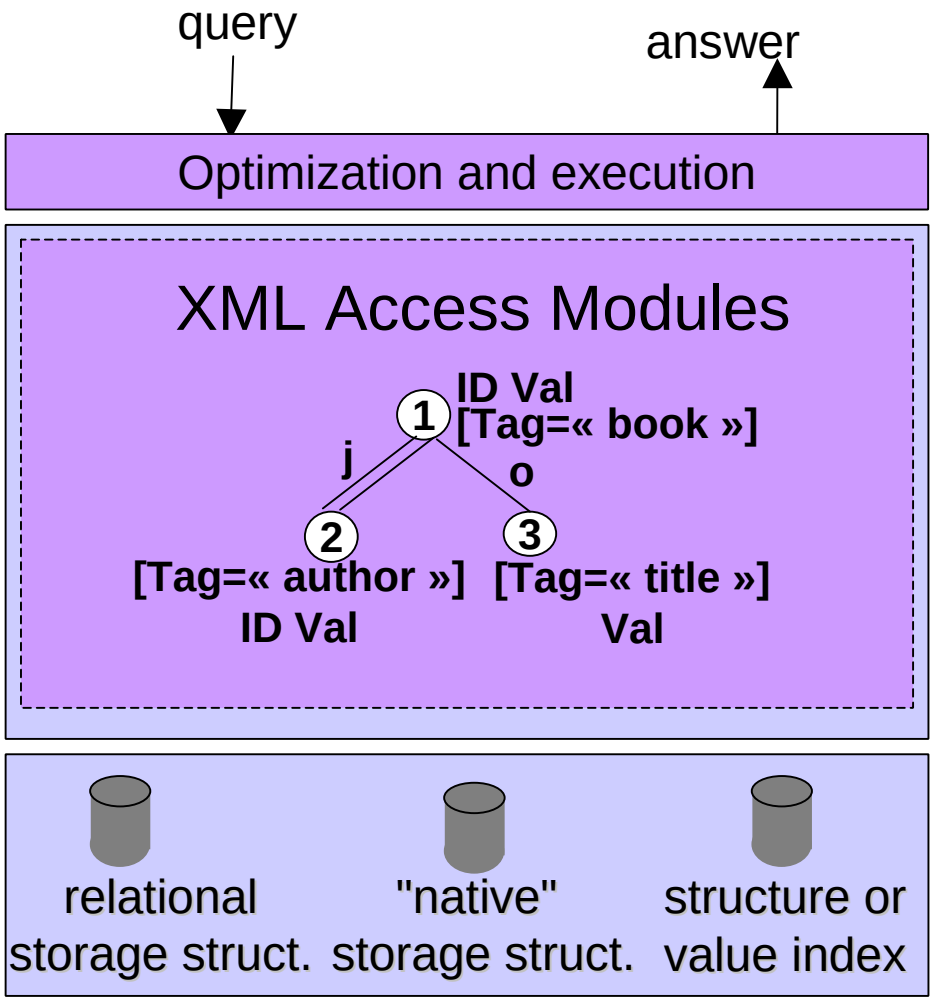
- Many existing storage and indexing models for XML
- Different applications and data sets call for different storage structures

Query processing on XML persistent stores



- Many existing storage and indexing models for XML
- Different applications and data sets call for different storage structures
- Rewriting the optimizer for every new storage model not an option
- We need:
  - High-level language for describing disk-resident storage structures
  - Algorithms for query answering
  - Physical data independence

Query processing on XML persistent stores

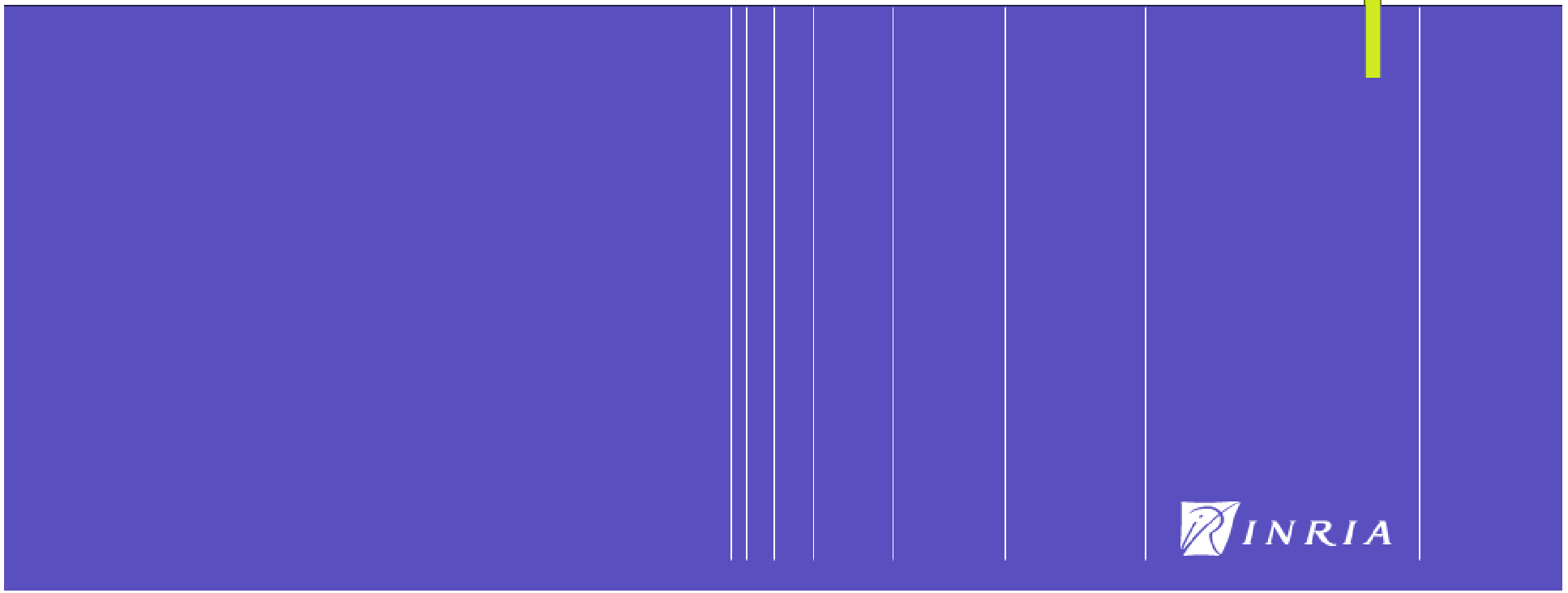


- Many existing storage and indexing models for XML
- Different applications and data sets call for different storage structures
- Rewriting the optimizer for every new storage model not an option
- We need:
  - High-level language for describing disk-resident storage structures
  - Algorithms for query answering
  - Physical data independence

Related problems

- XML query cache
  - Several data fragments have been loaded in cache by previous queries
  - Problem: answer a new query based on cache fragments ?
- XML database self-tuning
  - Given a document and some queries, which indexes/ materialized views to store to improve performance ?
- Local-as-view data integration
  - Several data sources, each storing part of a global dataset
  - Problem: answer a query over the global data by combining data sources ?

XML Access Modules (XAMs)



A language for XML materialized views and indexes

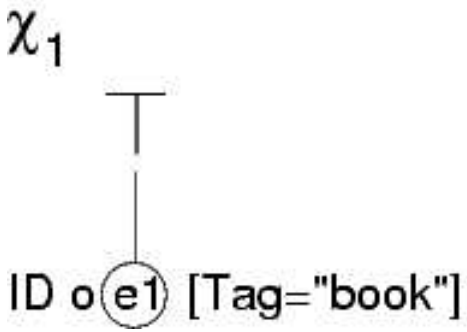
- Granularity
  - Small: values, persistent IDs
  - Large: full subtrees
- Organization
  - Trees seem natural
  - Nested tuples have clean algebraic foundations
- Specification
  - Value and structure conditions (~ selections)
  - Value and structure information (~ projections)

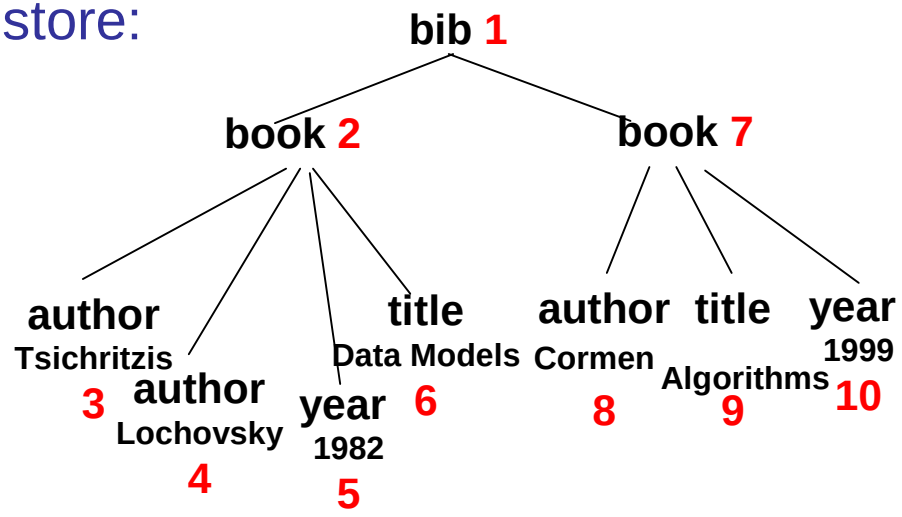
XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

For any node, the XAM may store:

- ID
- Tag
- Val
- Cont



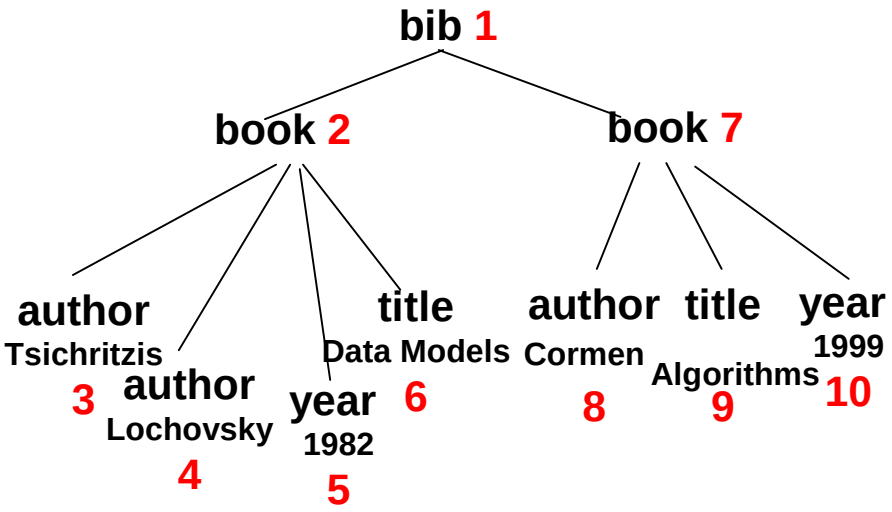


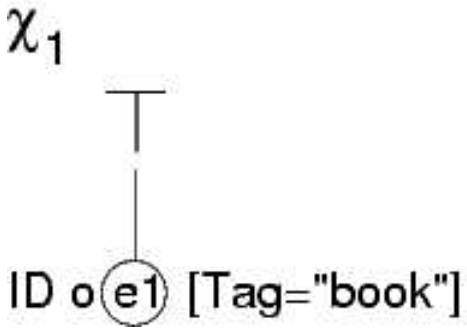
X1 tuples : [(2)
(7)]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

ID: i ID
- o order-preserving
- s structural
- n upward navigable
- u update resilient

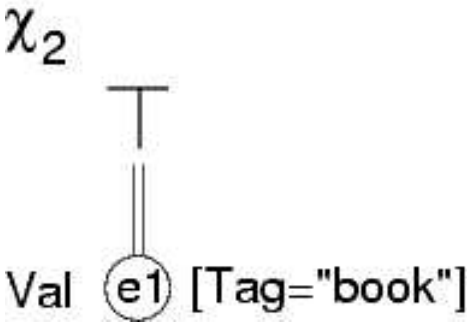


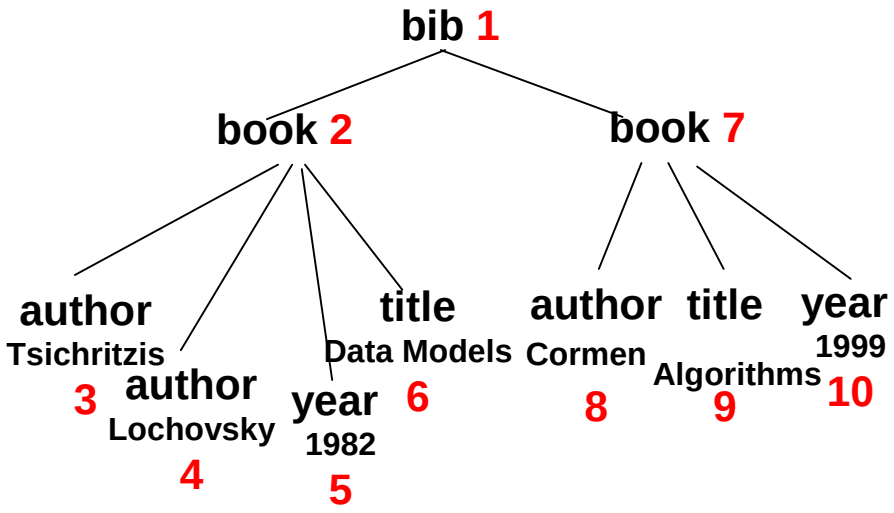


X1 tuples : [(2)
(7)]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

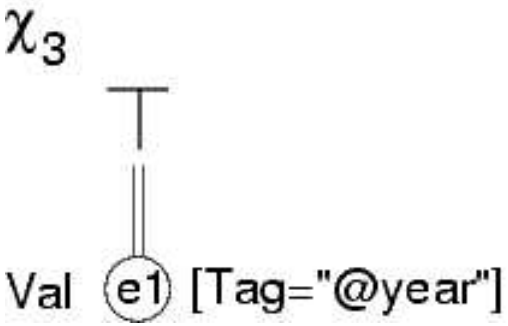


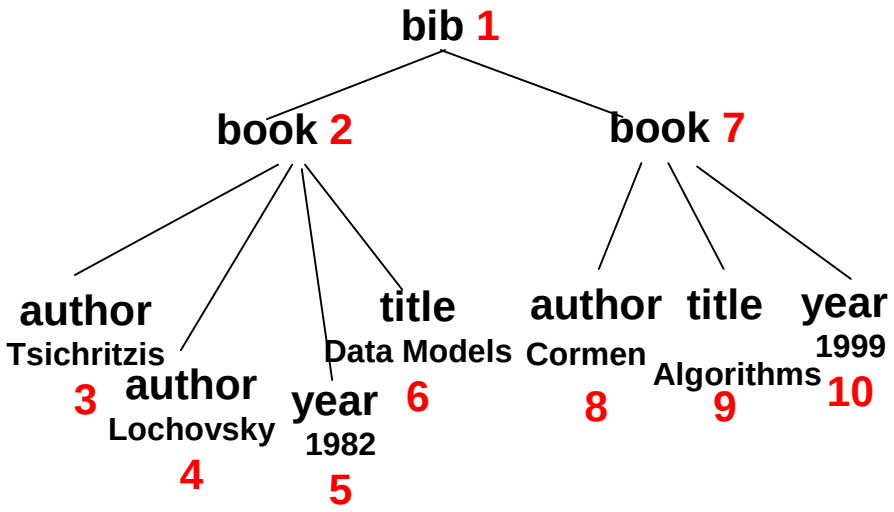


X2 tuples : [(⊥)
(⊥)]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

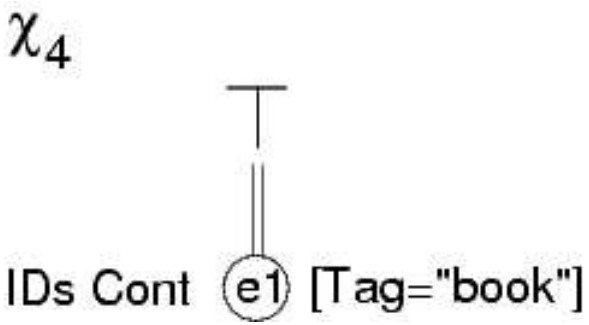


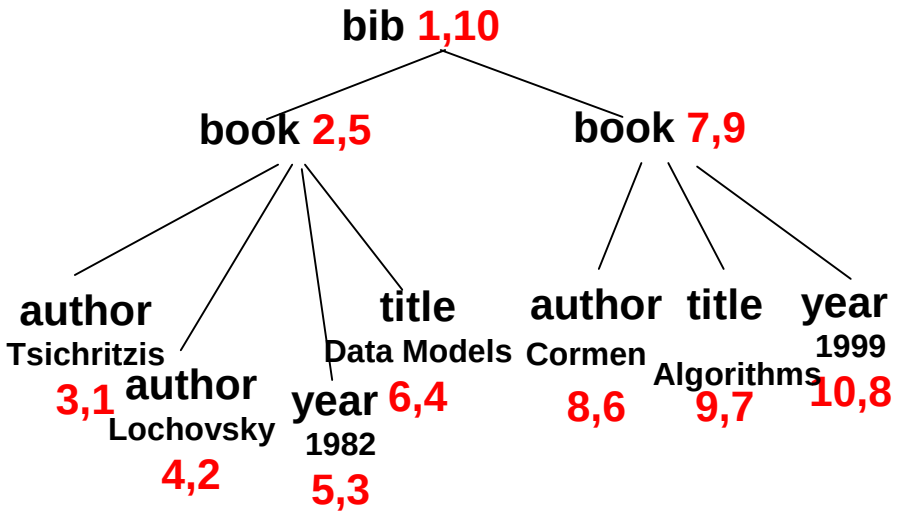


X3 tuples: [("1982")
("1999")]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

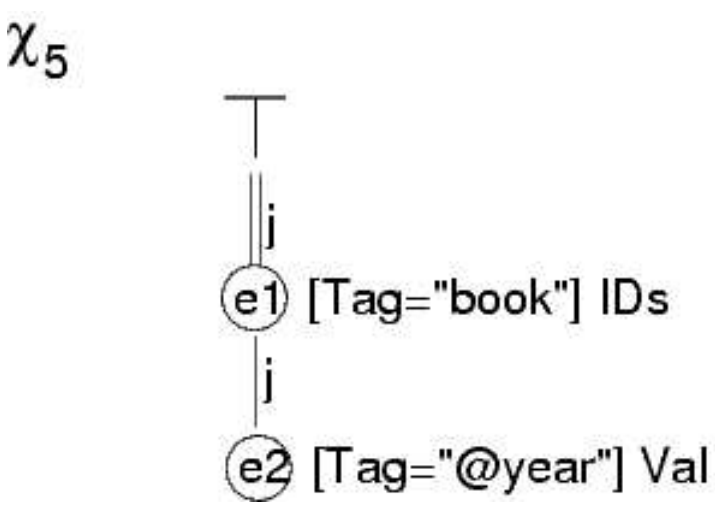


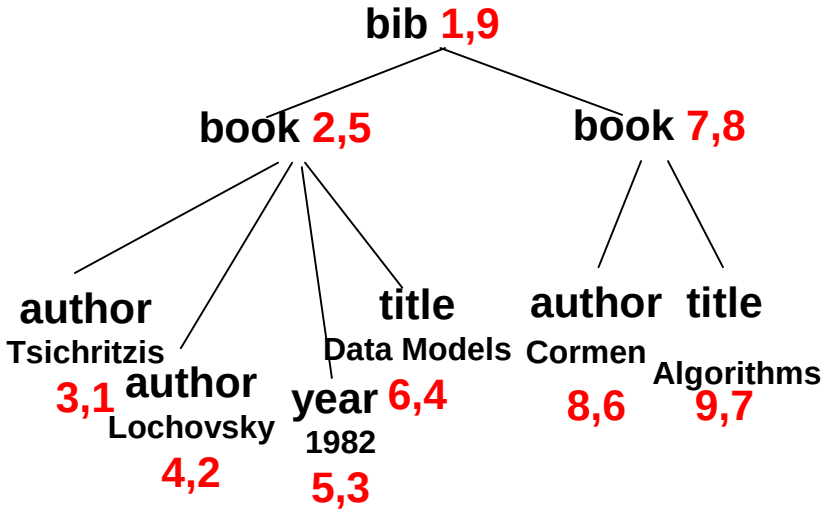


X4 tuples : [((2,5), <book year="1982"><author>Tsichritzis</author>
<author>Lochovsky</author> <title>Data Models</title>
</book>)
((7,9), <book year="1999"><author>Cormen</author><title>Algorithms</title>
</book>)]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

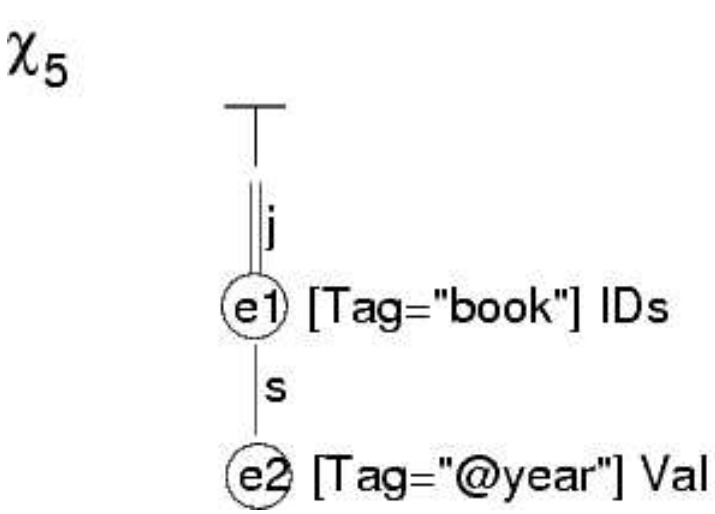


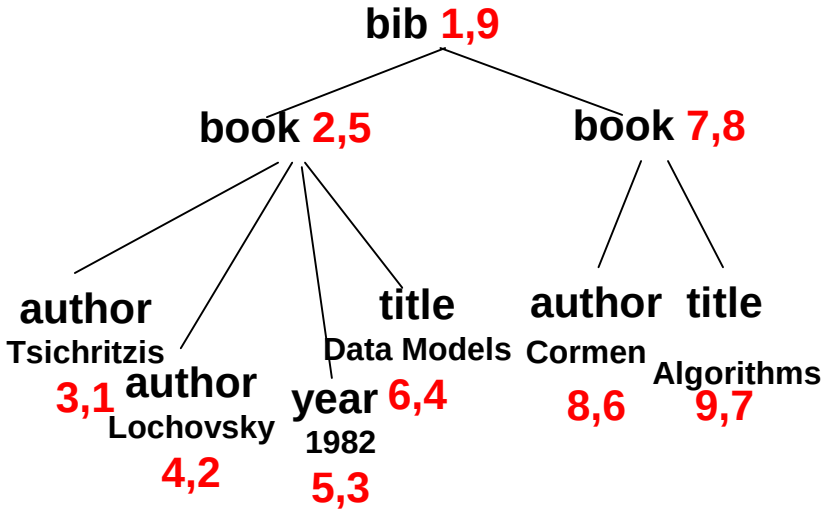


X5 tuples: [((2,5), "1982")]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

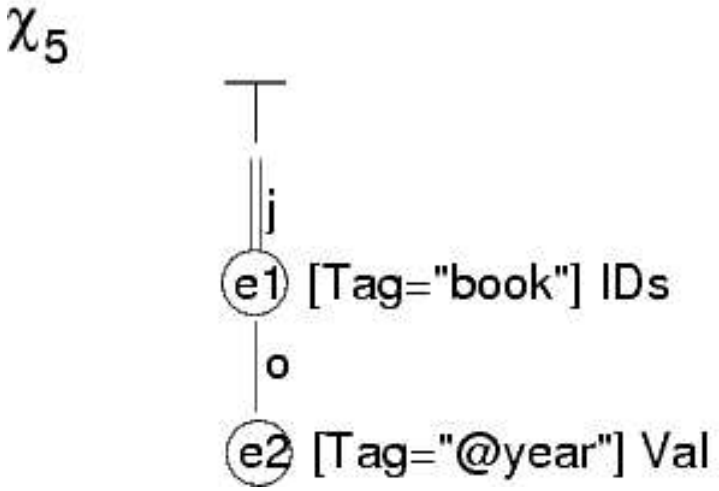


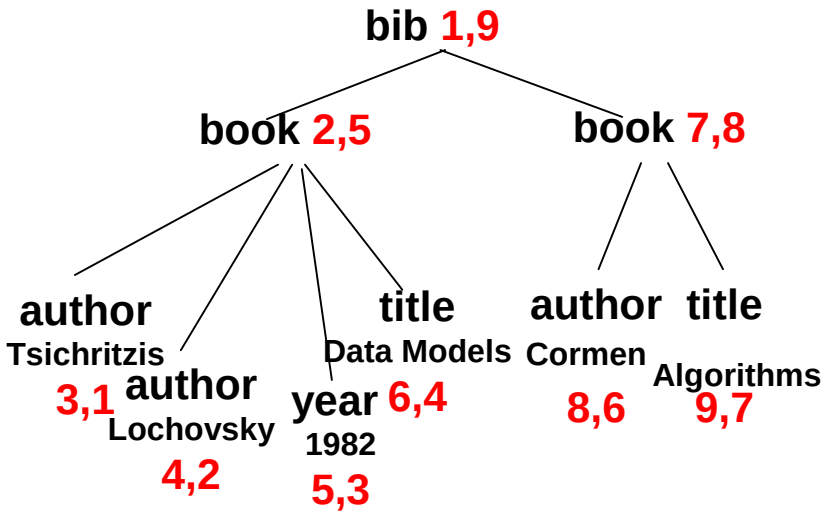


X5 tuples: [((2,5))]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

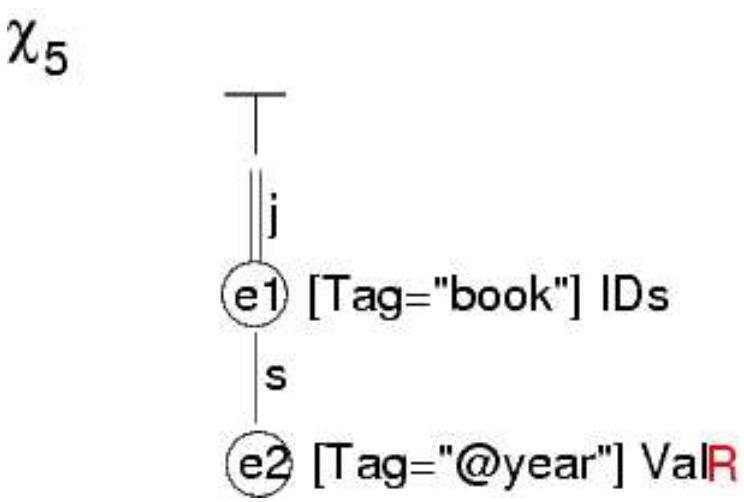


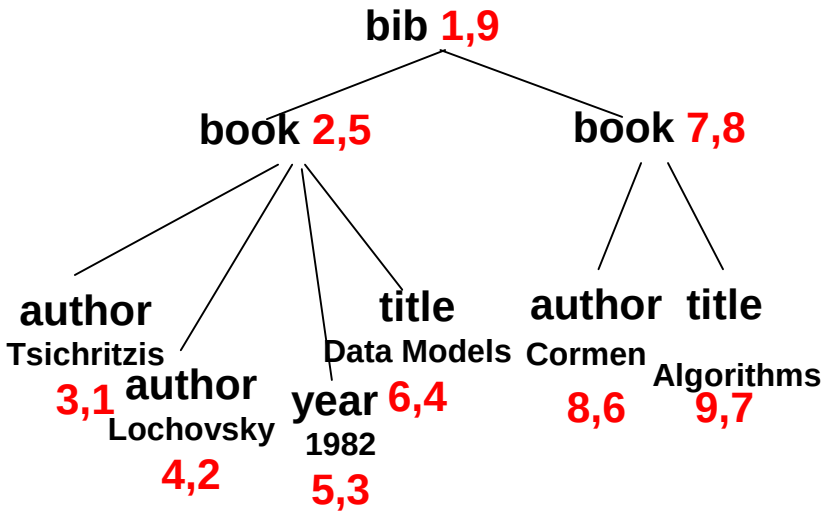


X5 tuples: [((2,5), "1982")
((7,9), ⊥)]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

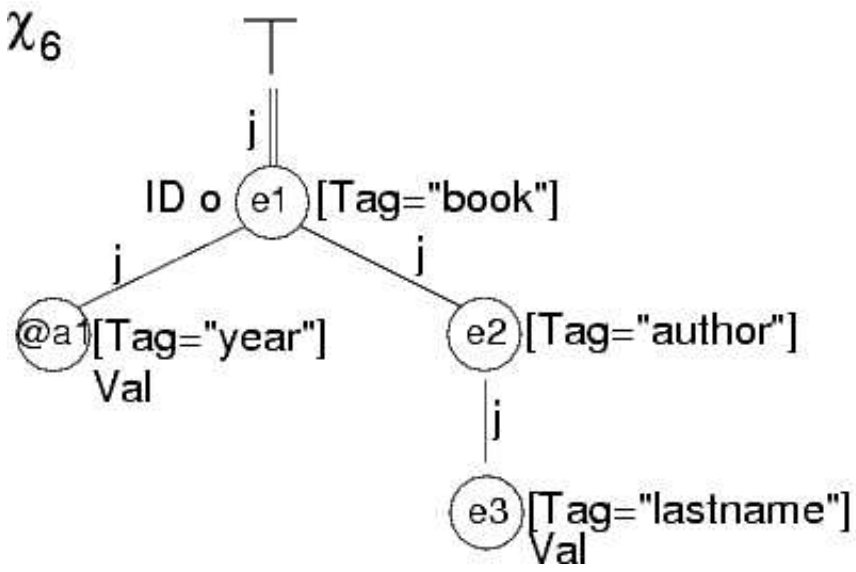


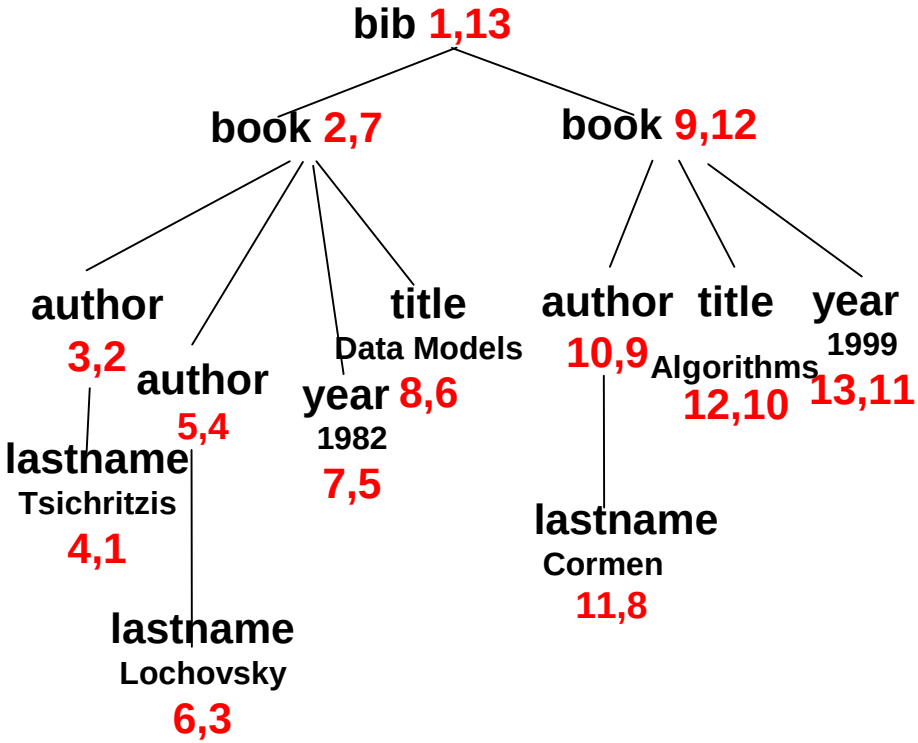


X5 contents: "1982" --> [((2,5), "1982")]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

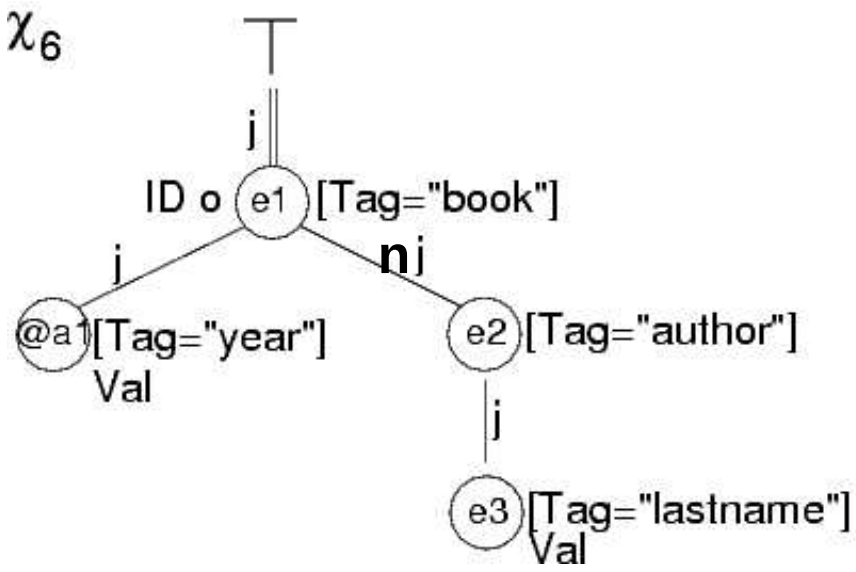


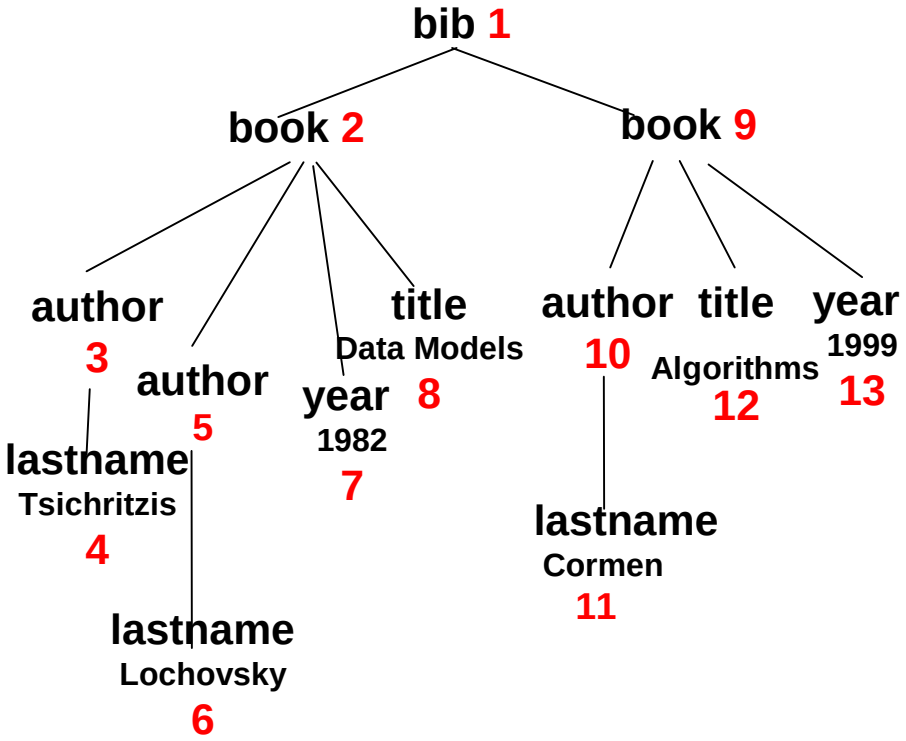


X6 tuples: [(2, "1982", "Tsichritzis")
(2, "1982", "Lochovsky")
(9, "1999", "Cormen")]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

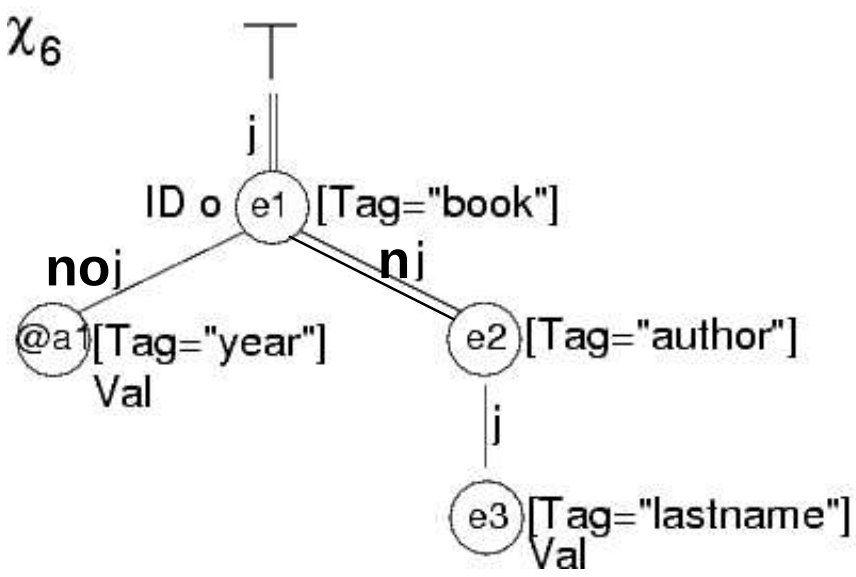


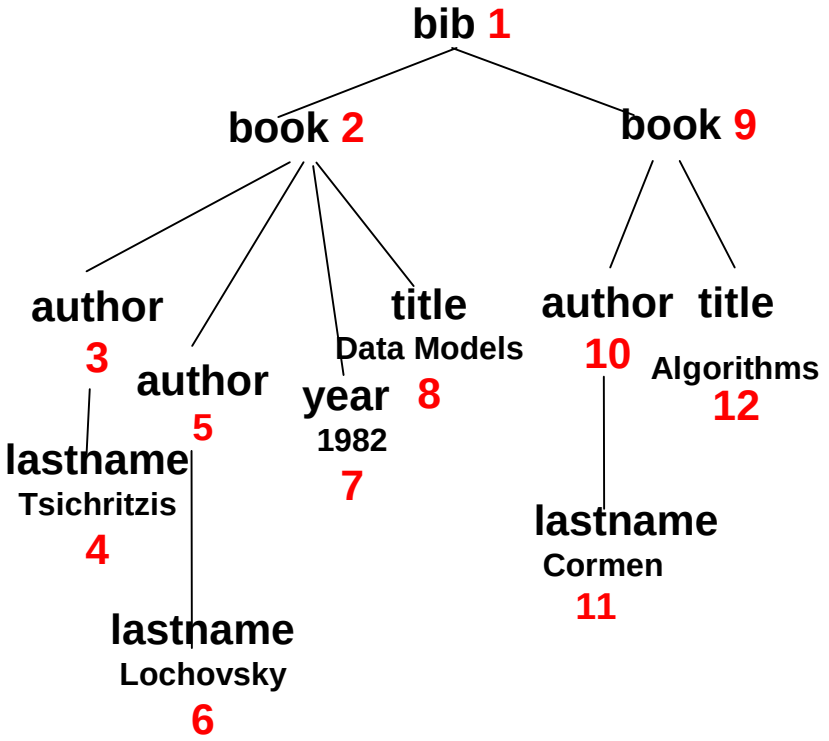


X6 tuples: [(2, "1982", [("Tsichritzis"), ("Lochovsky")])
(9, "1999", [("Cormen")])]

XML Access Modules (XAMs)

Tree-based language; nested tuple semantics

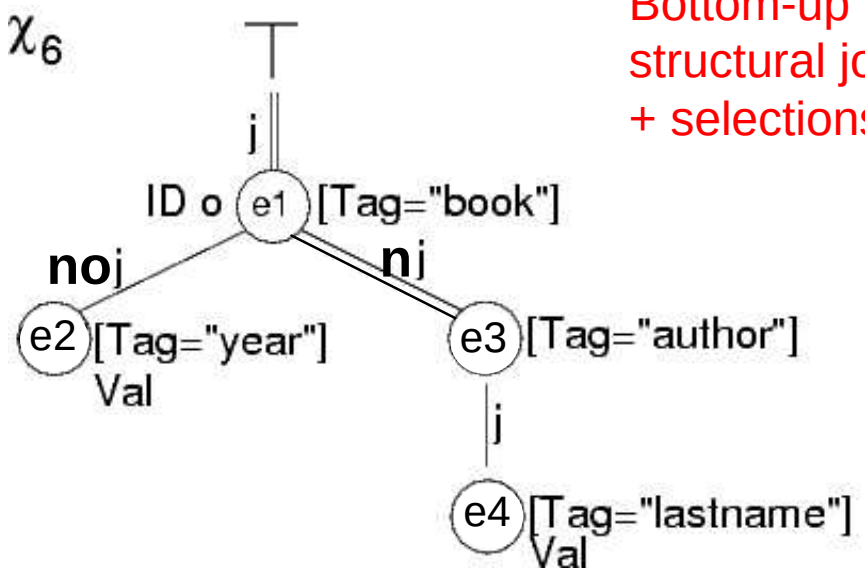




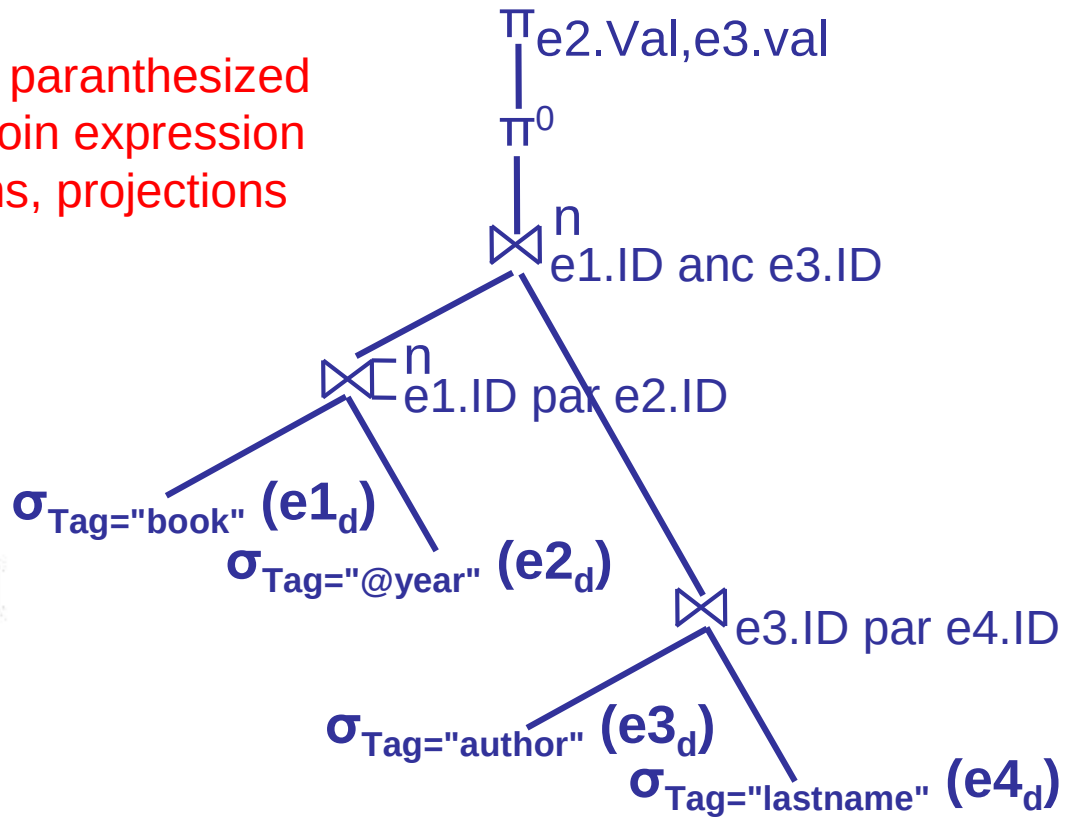
X6 tuples: [(2, [("1982")], [("Tsichritzis") ("Lochovsky")])
(9, [ ], [("Cormen")])]

XAM semantics

For a document d, consider basic relation e_d(ID,Tag,Val,Cont)



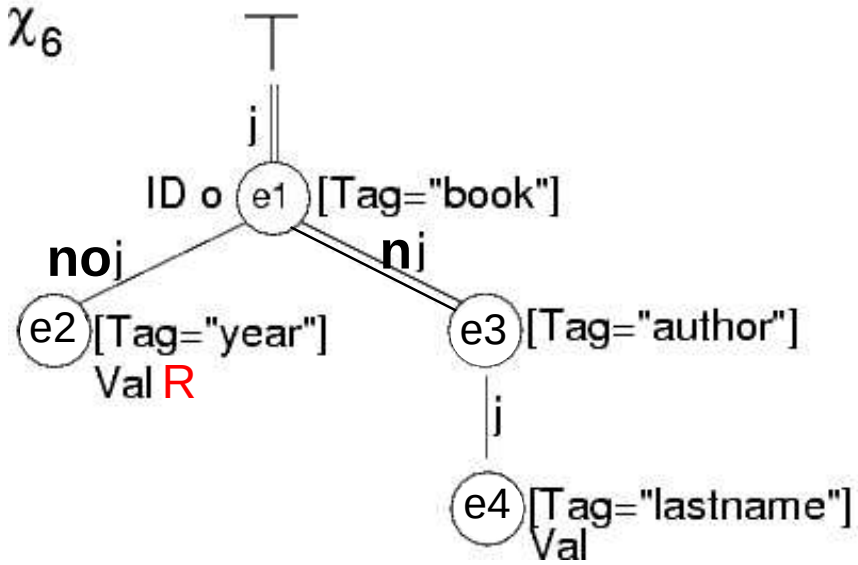
Bottom-up paranthesized structural join expression + selections, projections



XAM semantics with access restrictions

Let X_0 be the XAM obtained from X by removing all R annotations

Let t_0 be a tuple of bindings for the R-annotated attributes



Content of X with bindings t_0 =

$\sigma_{Rattribs=t0}$(content of X_0)

XAM generality

Capture many of the XML fragmentation schemes previously proposed for storage and indexes

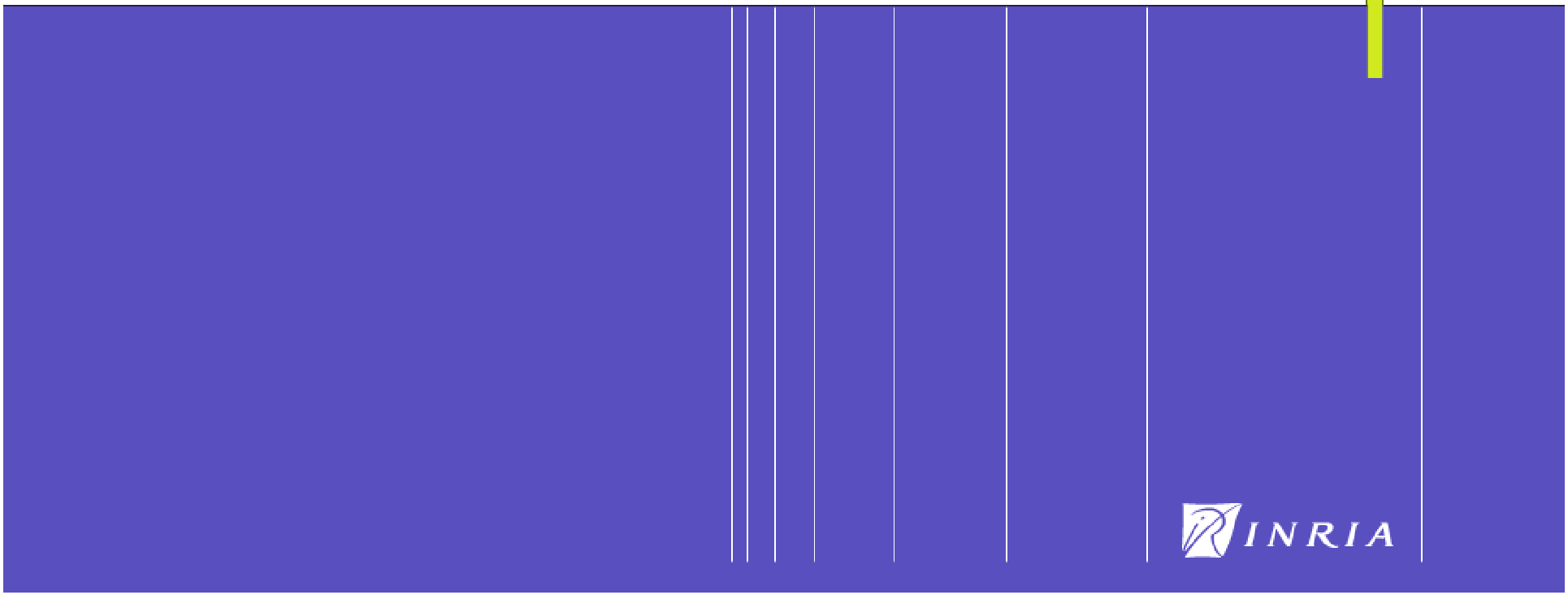
- Tag and path partitioning, "Shared", 1-index, F-index...

Also capture original nesting !!!

Do not capture

- Other navigation axes: "all a elements with their b siblings"
- Negation (antijoins): "all a elements without a b child"
- Value joins across unrelated elements
- Restructuring

Answering queries over XAMs



Problem statement

Input: a query Q and a set of XAMs X_1, X_2, ..., X_n

Output: all algebraic expressions $e(X_1, X_2, ..., X_n)$ such that

for any document d, $Q(d)=e(X_1, X_2, ..., X_n)(d)$

Algebraic expression ingredients:

scan(Xi), σ,π

$\bowtie_{par/anc}$ $\bowtie_{pred}$ $\bowtie^{n}_{par/anc}$ $\bowtie^{n}_{pred}$ $\leftthreetimes_{par/anc}$ $\leftthreetimes_{pred}$ $\leftthreetimes^{n}_{par/anc}$ $\leftthreetimes^{n}_{pred}$ $\ltimes_{par/anc}$ $\ltimes_{pred}$

navigation in XML serialized trees (Cont attributes)

Problem statement

Input: a query Q and a set of XAMs X_1, X_2, ..., X_n

Output: all algebraic expressions $e(X_1, X_2, ..., X_n)$ such that

for any document d, $Q(d)=e(X_1, X_2, ..., X_n)(d)$

Remark: if $e1=X1 \bowtie X2$ is equivalent to Q, then also

$e2=X2 \bowtie X1$ is equivalent to Q

Problem statement

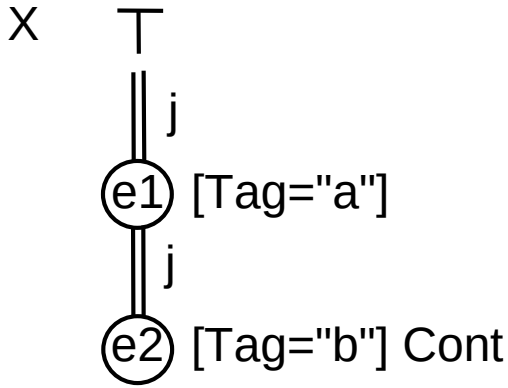
Input: a query Q and a set of XAMs X_1, X_2, ..., X_n

Output: all algebraic expressions $e(X_1, X_2, ..., X_n)$

(up to algebraic equivalence)

such that for any document d, $Q(d)=e(X_1, X_2, ..., X_n)(d)$

Remark: consider the XAM X and queries Q1= //a//b, Q2=//b



X can be used for Q1 in general.

X can also be used for Q2 if all b elements have an a ancestor

Problem statement

Input: a query Q and a set of XAMs $X_1, X_2, ..., X_n$

structural constraints on the document used by Q

Output: all algebraic expressions $e(X_1, X_2, ..., X_n)$

(up to algebraic equivalence)

such that for any constr'd. doc. d, $Q(d)=e(X_1, X_2, ..., X_n)(d)$

Remark: we are interested in plans that can actually be translated into executable ones

$\bowtie_{par/anc}$ $\bowtie\!\!\!\!\sqsubset_{par/anc}$ $\ltimes_{par/anc}$ $\bowtie^{n}_{par/anc}$ $\bowtie\!\!\!\!\sqsubset^{n}_{par/anc}$ only on ID s,n

Navigation only on Cont attributes

Problem statement

Input: a query Q and a set of XAMs X_1, X_2, ..., X_n

structural constraints on the document used by Q

Output: all valid algebraic expressions $e(X_1, X_2, ..., X_n)$

(up to algebraic equivalence)

such that for any constr'd. doc. d, $Q(d)=e(X_1, X_2, ..., X_n)(d)$

Rewriting algorithm outline

1. Construct algebraic expressions from Q

   Downward XPath leads to XAM-like algebraic expressions

   Downward XQuery leads to a join over several XAM-like algebraic expressions XQ_1, XQ_2, ..., XQ_m.

2. Inject constraint information into Q and X_1, X_2, ..., X_n.

3. For each XQ_i

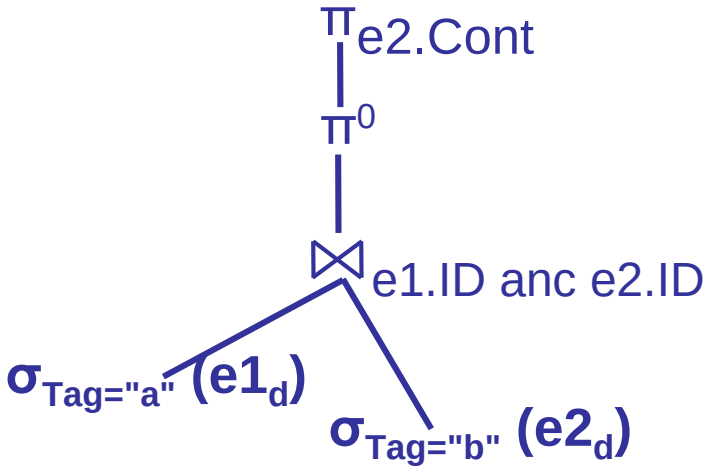
   3.1 Construct gradually larger algebraic expressions starting from $\pi(Scan(X_j))$, until current expr. is contained in XQ_i.

   3.2 Cover XQ_i with unions of contained rewritings

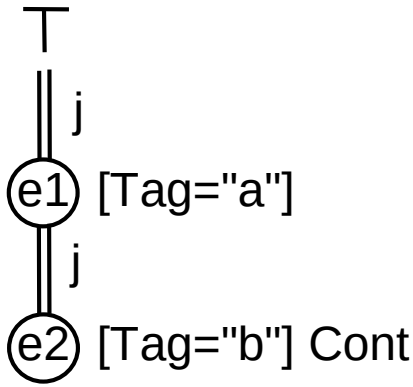
4. Combine all rewritings for XQ_1, XQ_2, ..., XQ_m via joins

From XPath to algebra

//a//b

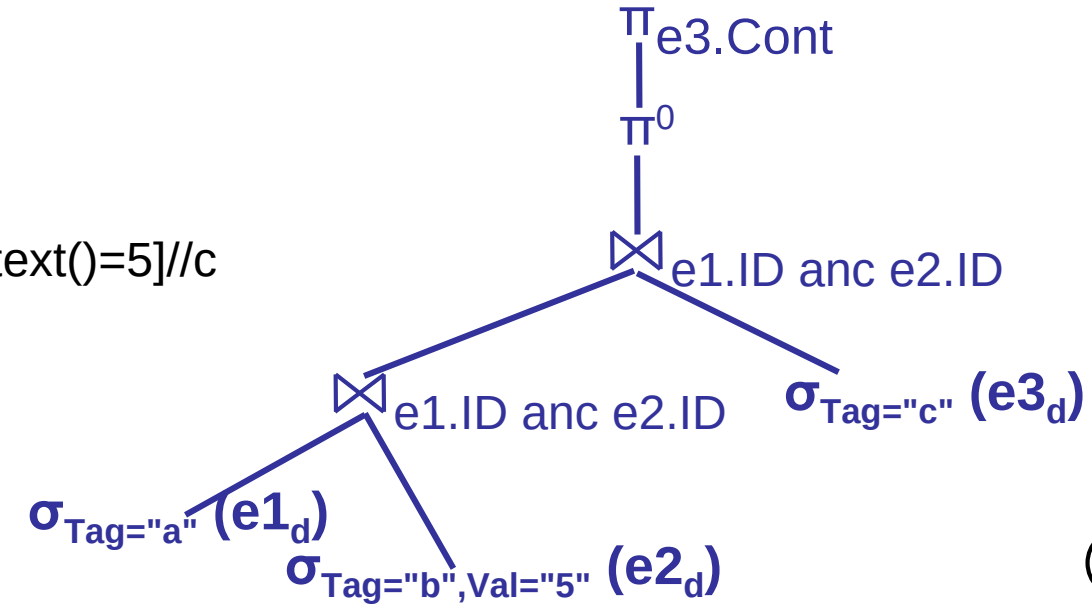


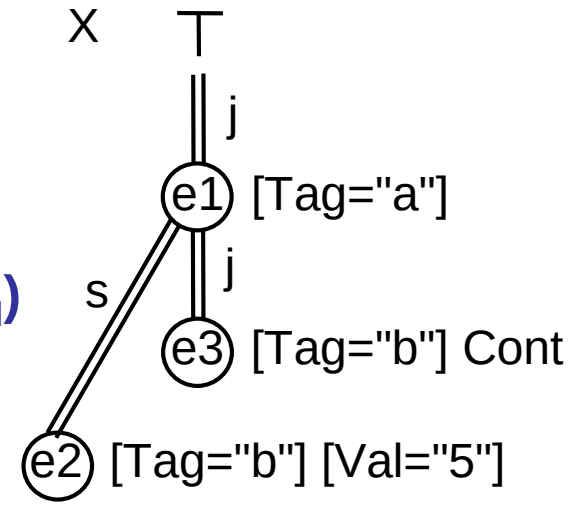
X



From XPath to algebra

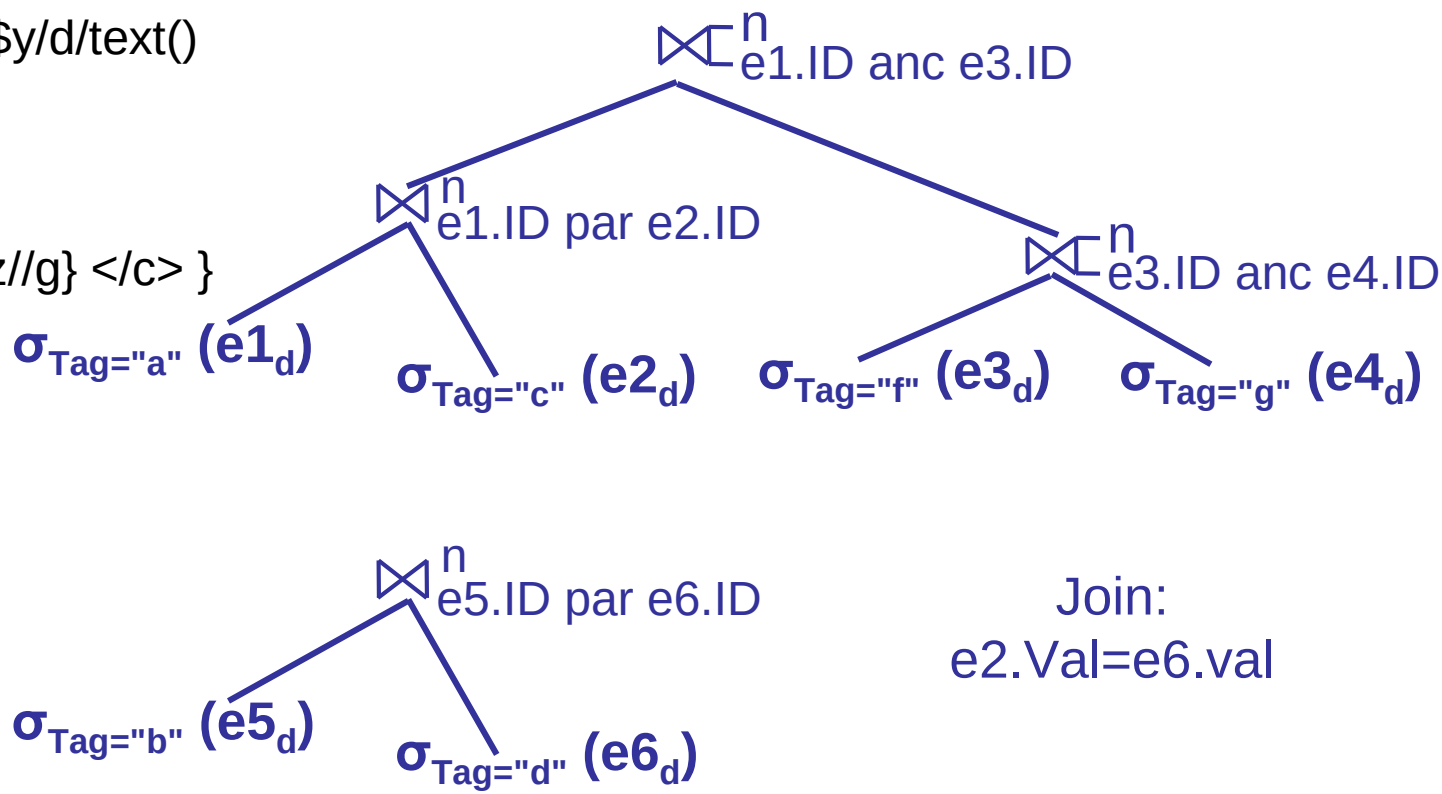
//a[//b/text()=5]//c





From XQuery to algebra

```
for $x in //a, $y in //b
where $x/c/text()=$y/d/text()
return
<new>
     { for $z in $x//f
       return <c> {$z//g} </c> }
</new>
```



Injecting constraints in views and queries

Annotate views and query with information allowing to infer which view nodes may bring information for which query node

E.g. //*[inproceedings] is the same as //article

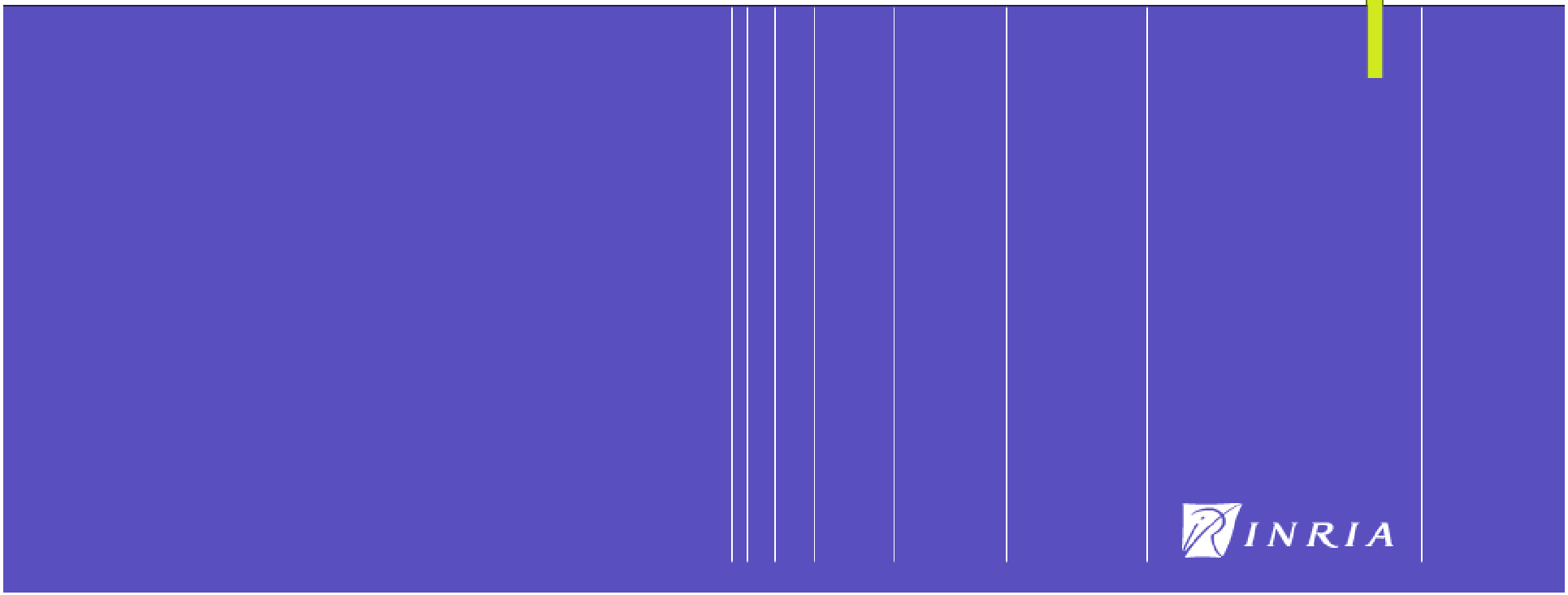
//*[inproceedings] is the same as //*[booktitle]

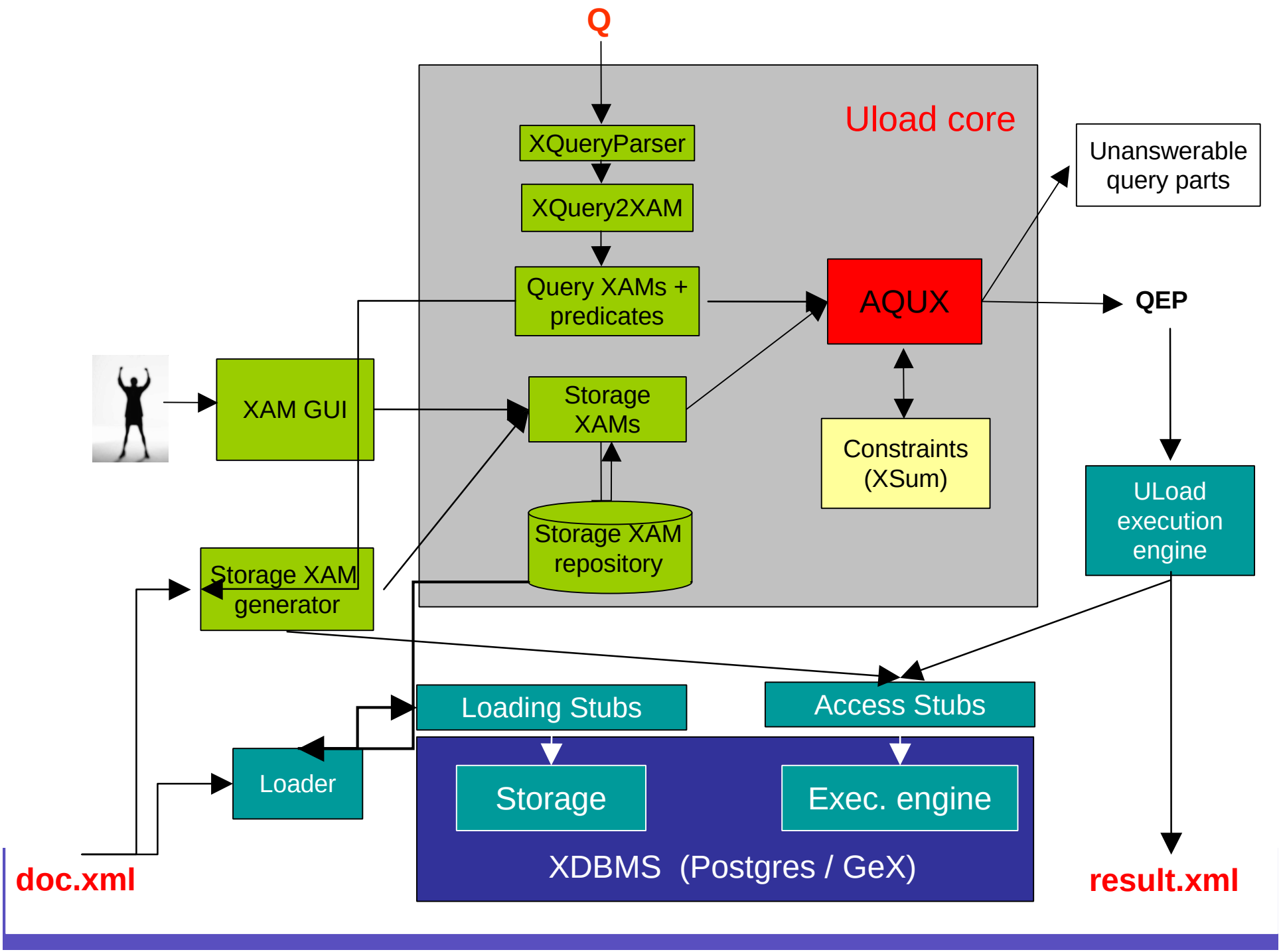
We used enhanced DataGuides as constraints. Schemas also apply.

In general, constraints allow to:

- Find more rewritings
- Avoid empty-result rewritings
- Find more efficient algebraic rewritings

ULoad: a materialized view management tool based on XAMs




Q
Uload core
XQueryParser
XQuery2XAM
Query XAMs + predicates
AQUX
Unanswerable query parts
QEP
XAM GUI
Storage XAMs
Constraints (XSum)
Storage XAM repository
ULoad execution engine
Storage XAM generator
Loading Stubs
Access Stubs
Loader
Storage
Exec. engine
XDBMS (Postgres / GeX)
doc.xml
result.xml

ULoad prototype demonstration

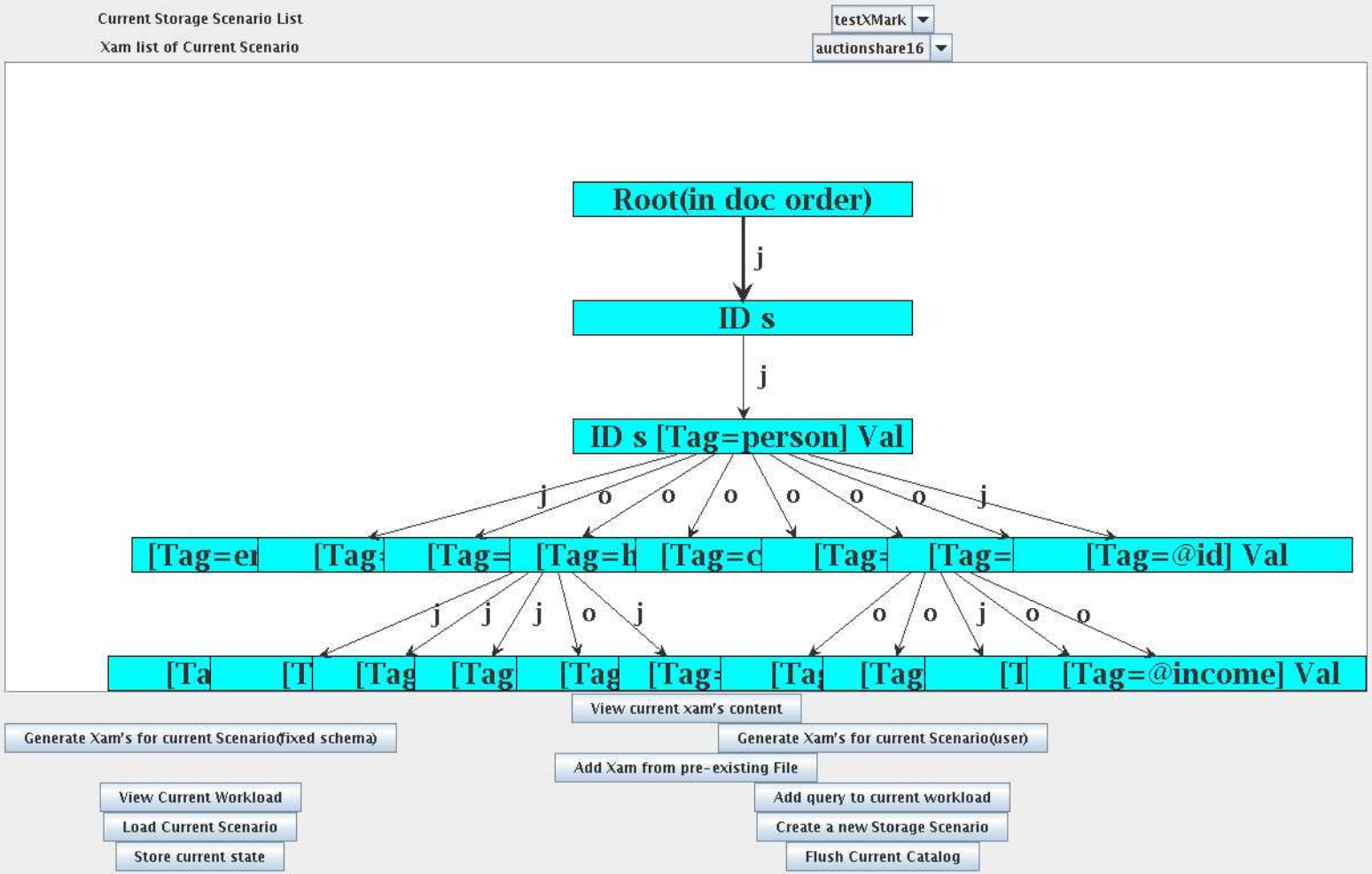
XML materialized view management for XQuery:

- Materialized view creation
- Data extraction & loading in native/relational repository
- Query answering over the materialized views
- Materialized view extraction from XQuery queries

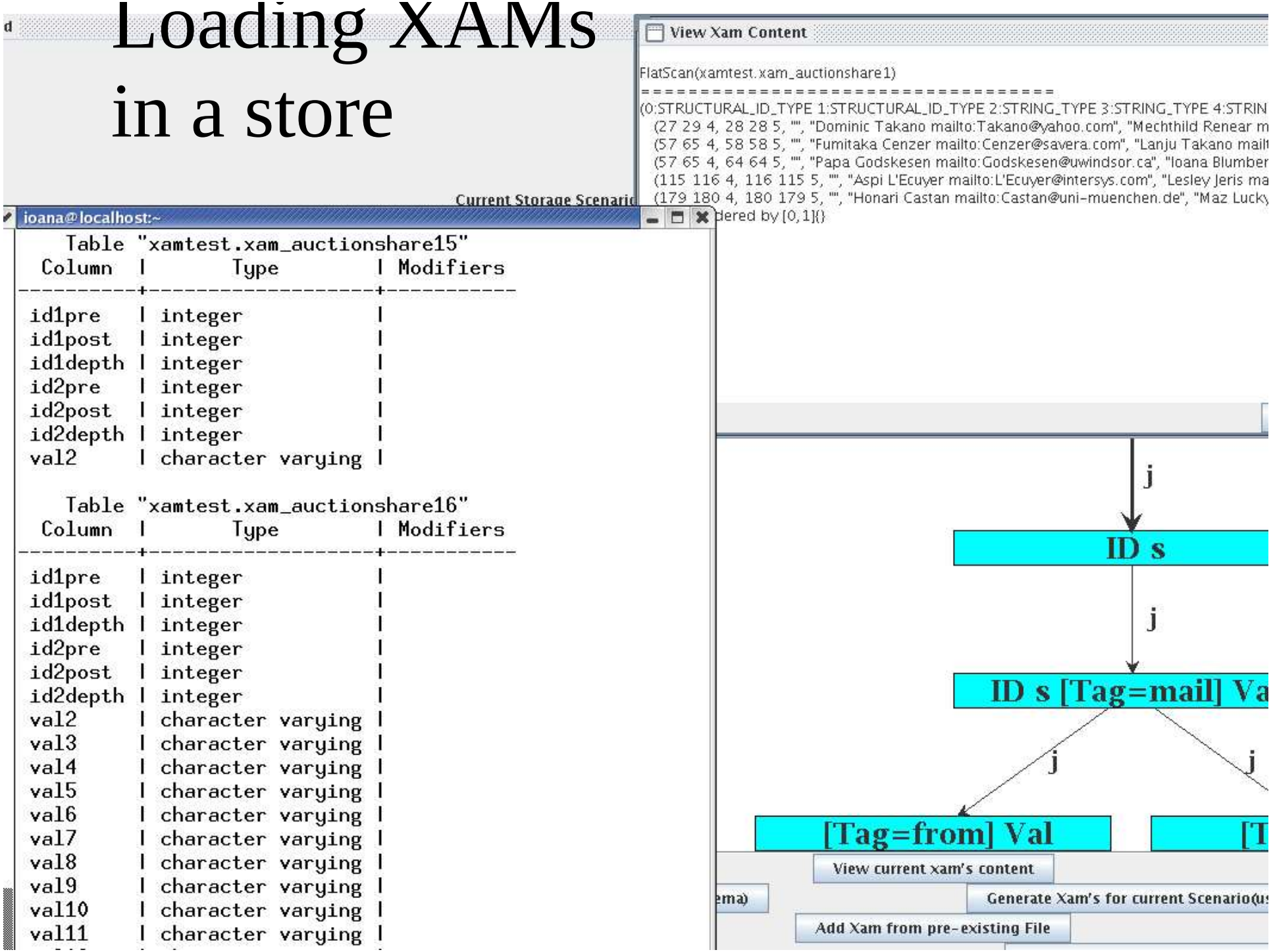
Also:

- Guidance in choosing views and writing queries: satisfiability / answerability tests

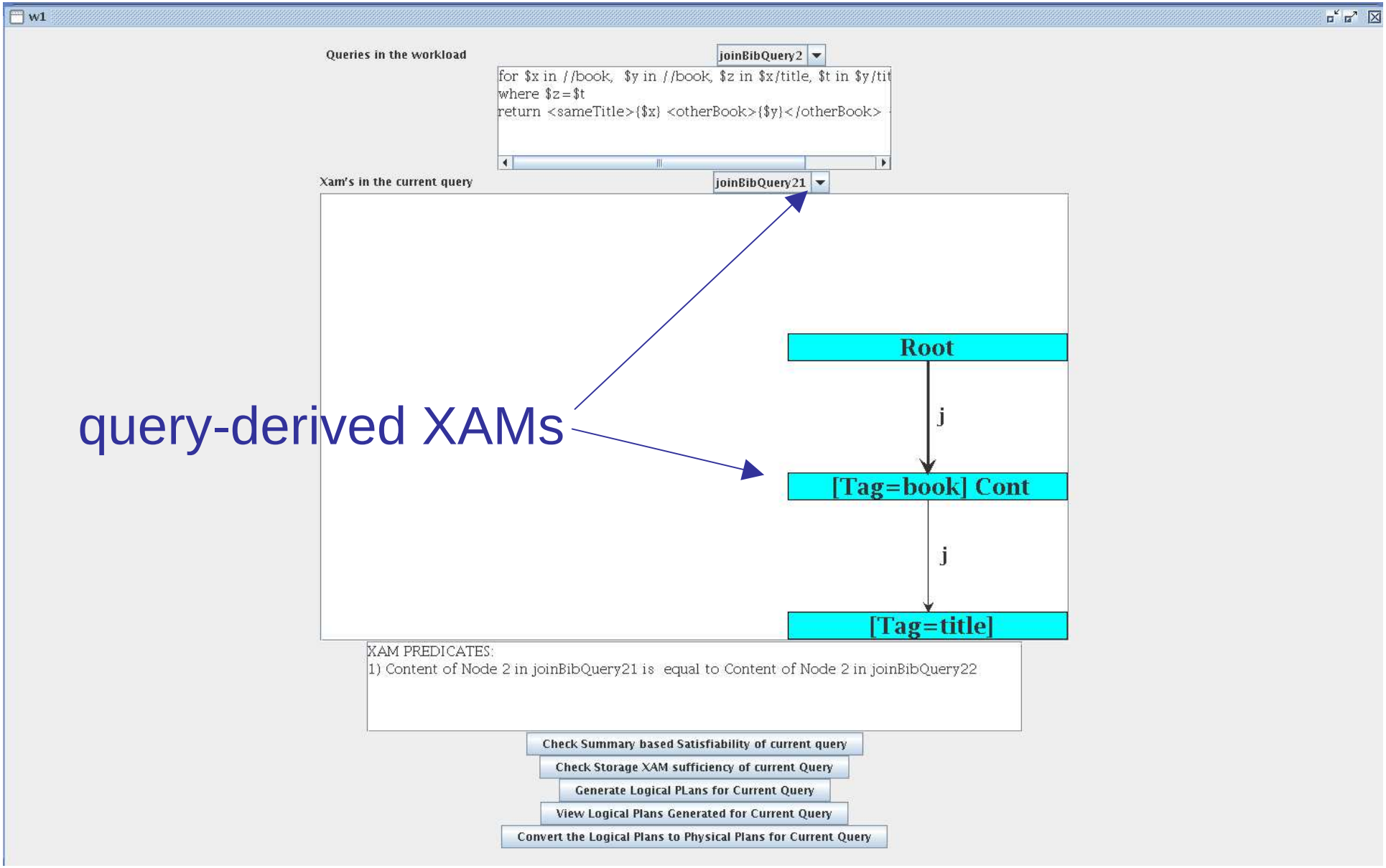
Formalism for describing complex XML materialized views: XAMs



Loading XAMs in a store

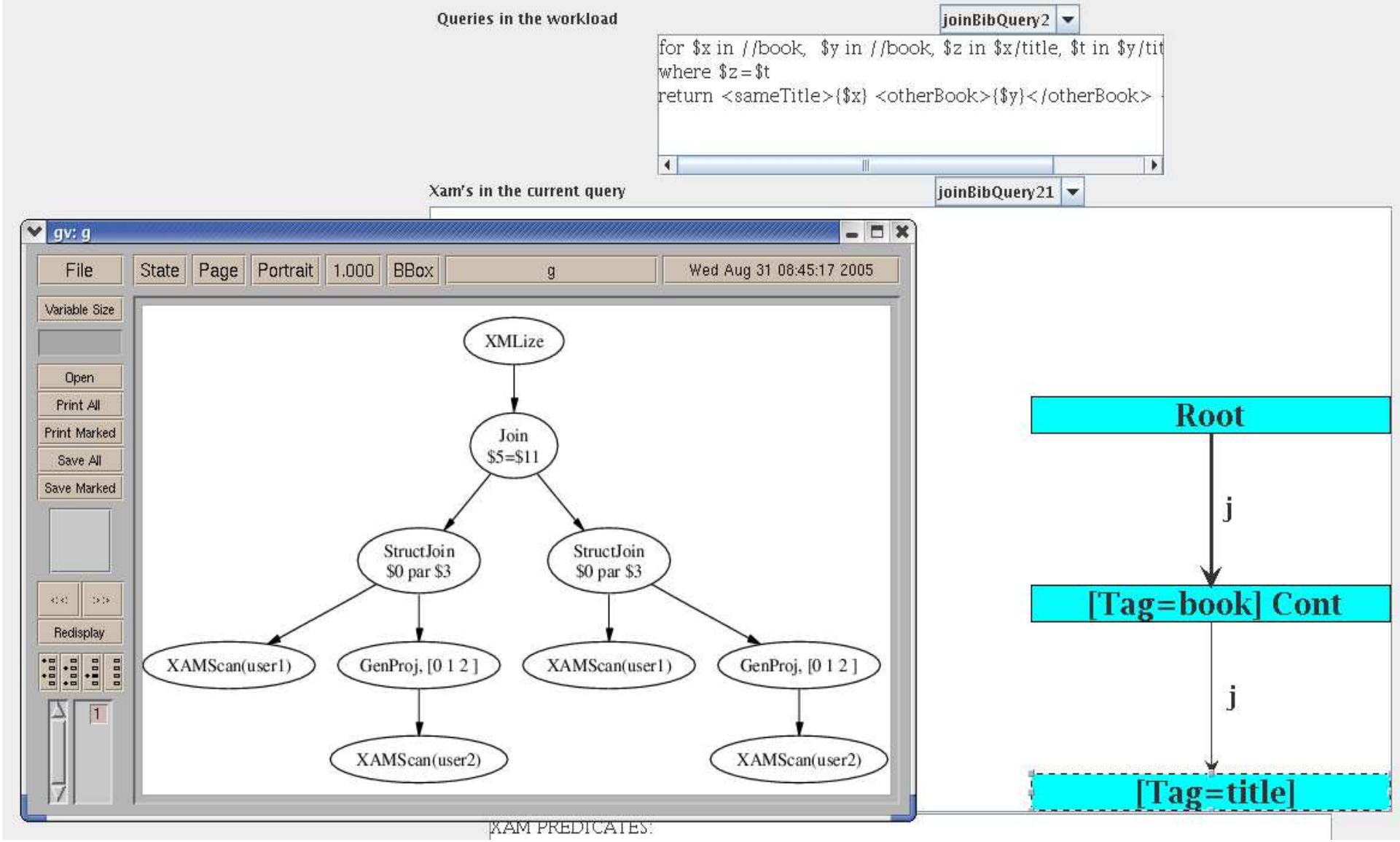


Querying a database of XAMs

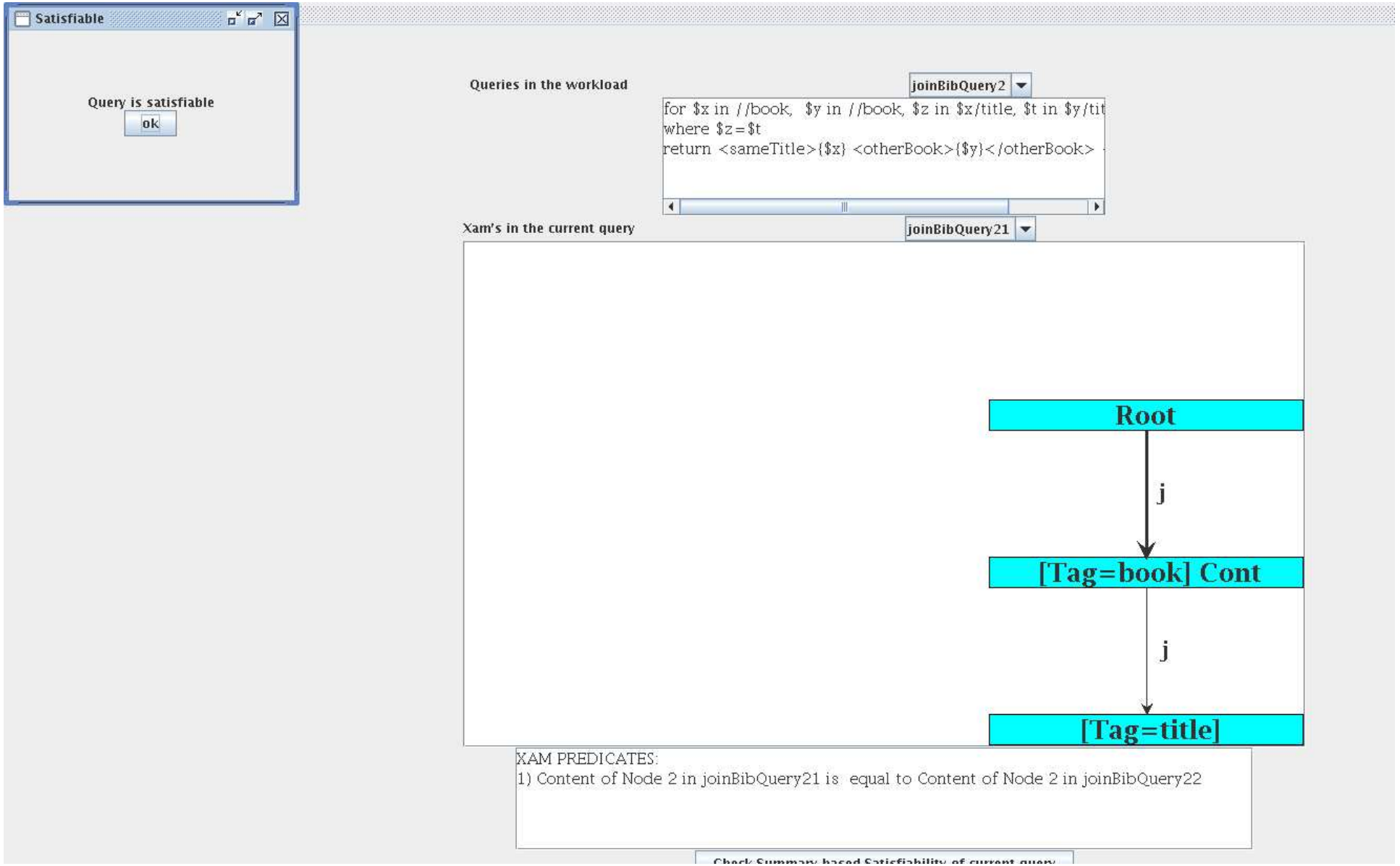


query-derived XAMs

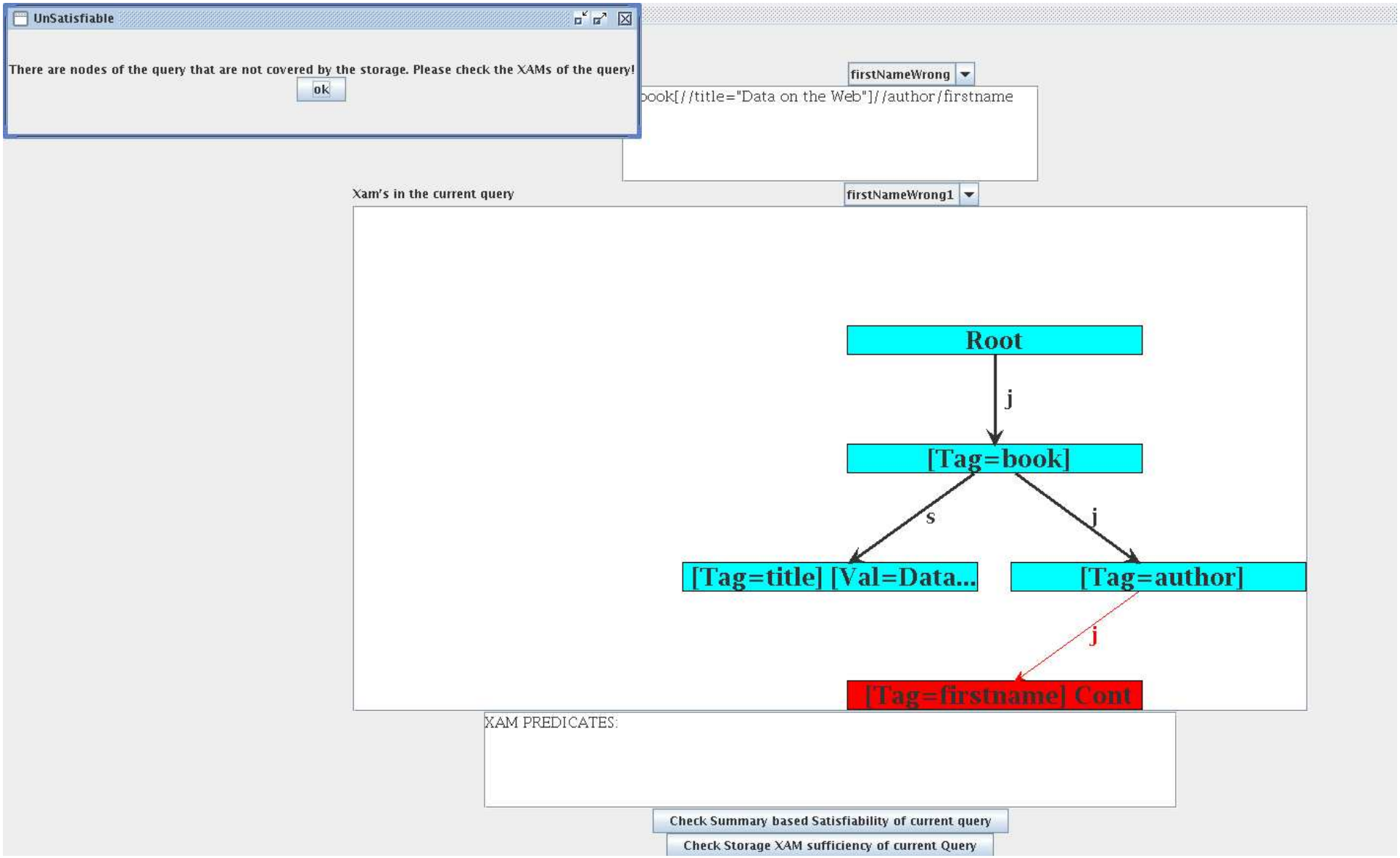
Logical query plans over XAMs



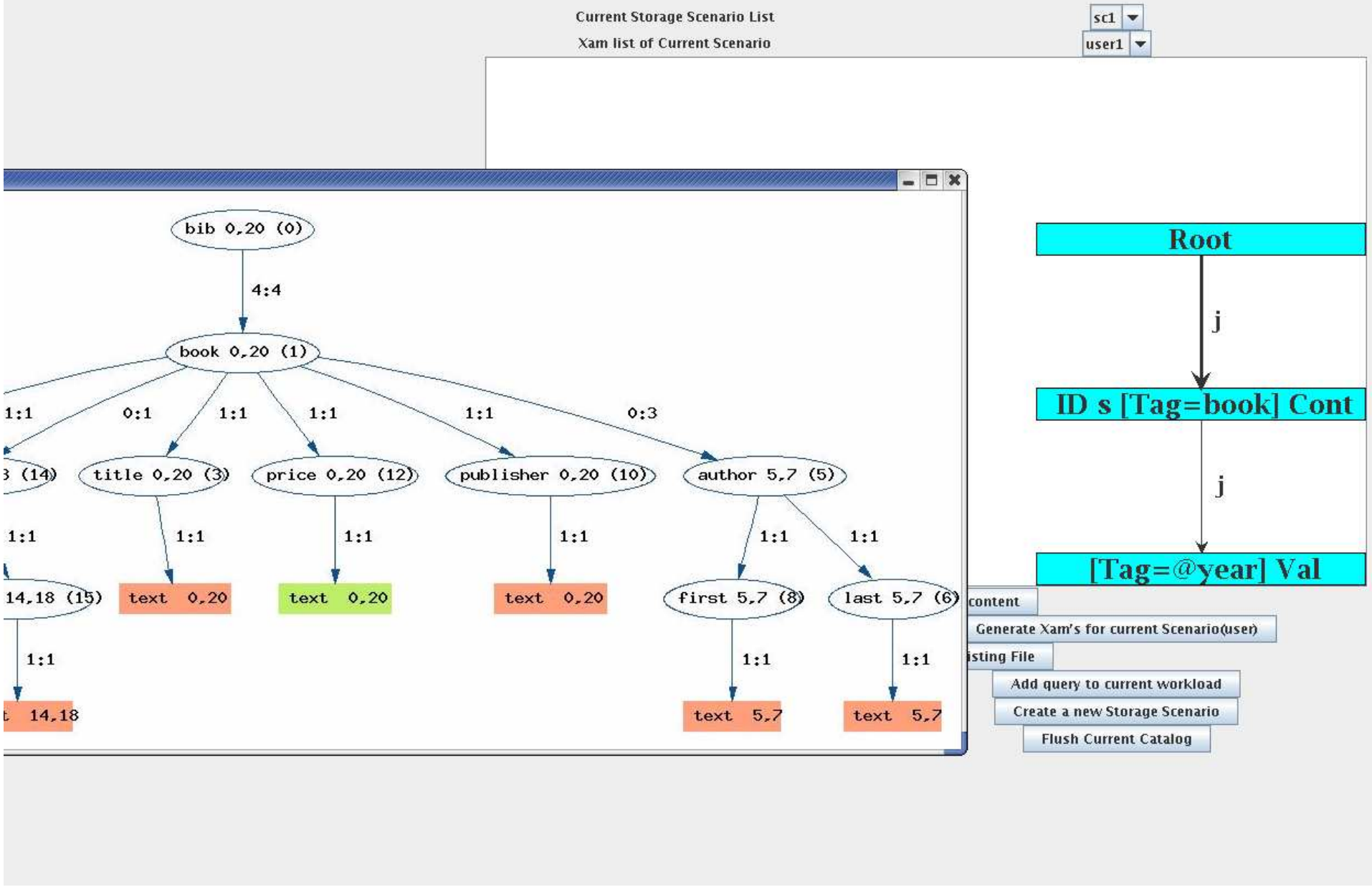
Testing query satisfiability



Testing query coverage by the stored XAMs



Behind the scene: structural constraints



More information

http://www-rocq.inria.fr/gemo/XAM

A.Arion, V.Benzaken and I. Manolescu. "XML Access Modules: Towards Physical Data Independence in XML Databases", XIME-P 2005

A.Arion, V.Benzaken, I. Manolescu, R.Vijay. "ULoad: choosing the storage for your XML application", VLDB 2005 (demo)

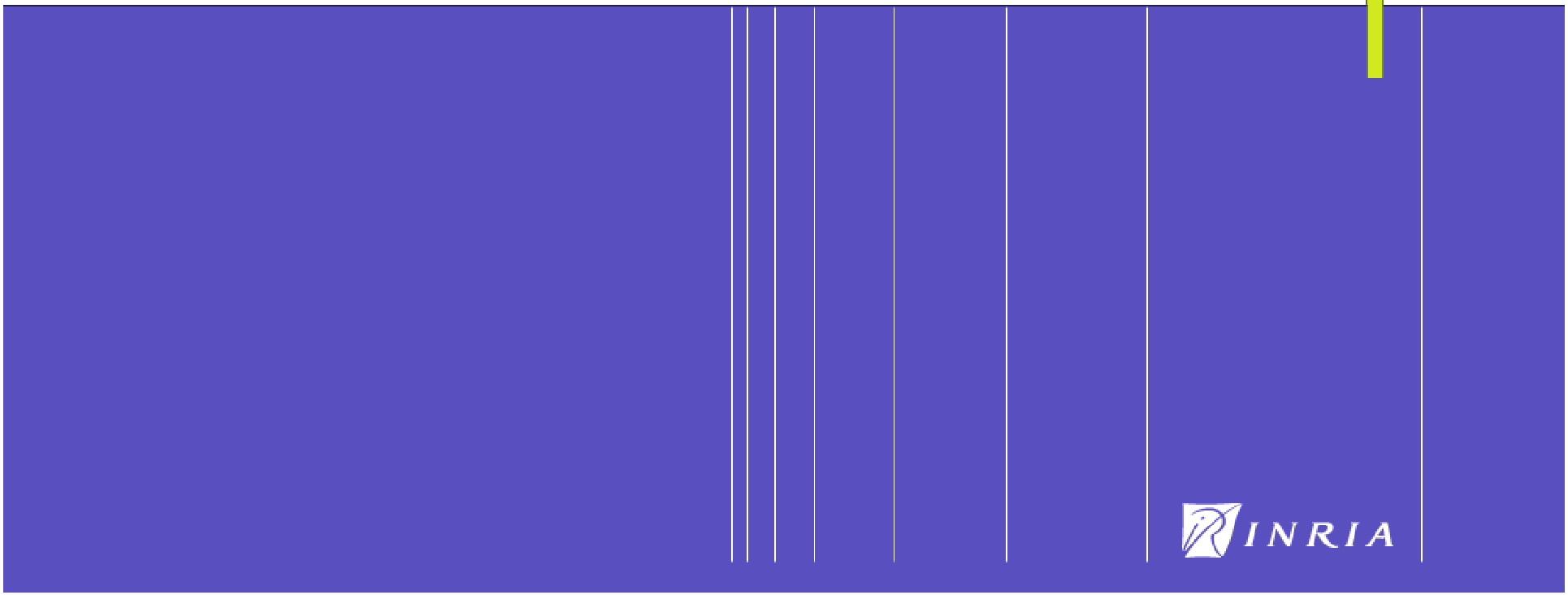
Tech. report upcoming

Perspectives

XAMS et ULoad:

- formaliser l'inclusion de requetes et la re-ecriture a base de XAMs (completude, complexite...)
- connexion avec l'environnement de requetes CDuce

Related works



Related works (1)

Long history of storage and indexing schemes

- Also known as "shredding" strategies, path indexes
- Shanmugasundaram et al. 1999, Benedikt et al. 2001, Kaushik et al. 2002, ...
  SQL/XML published in 2003

XPath containment and equivalence

- Deutsch and Tannen, 2001 and later; based on chase
- Suciu; Schwentick, Gottlob and Segoufin; Ozsoyoglu...
- We provide a practical method for XPath rewriting under specific constraints (easier !)

Related works (2)

Tree pattern minimization

- Amer-Yahia and Srivastava, 2001. Different containment,
- no rewriting

XQuery containment

- Halevy et al. 2004, different notion of containment, no constraints

XPath rewriting with materialized views

- Weak path usage. Balmin et al, VLDB 2004

Algebraic minimization of XQuery

- Deutsch and Papakonstantinou, VLDB 2004