University of Washington
Department of Computer Science & Engineering

Managing XML and Semistructured Data

Lecture 5: Query Languages - Lorel and UnQL

Prof. Dan Suciu

Spring 2001

In this lecture

- A core query language
- Lorel
- UnQL

Resources:

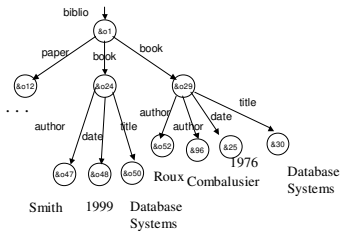
UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion Buneman, Fernandez, Suciu. VLDBJ 2000

The Lorel Query Language for Semistructured Data by Abiteboul, Quass, McHugh, Widom, Wiener, in International Journal on Digital Libraries, 1997.

A Core Query Language

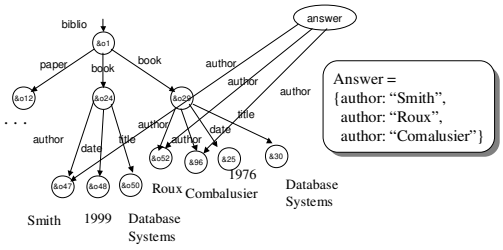
Will illustrate with:

DB =



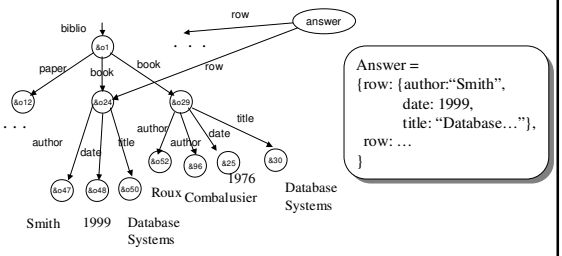
Query 1:

```
SELECT author: X
FROM biblio.book.author X
```



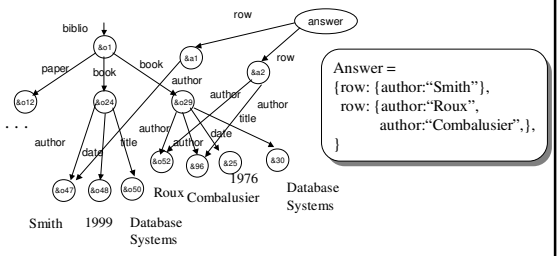
Query 2:

```
SELECT row: X
FROM biblio._ X
WHERE “Smith” in X.author
```



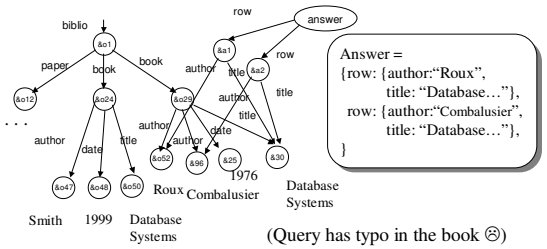
Query 3:

```
SELECT row: ( SELECT author: Y
                FROM X.author Y)
FROM biblio.book X
```



Query 4:

```
SELECT ( SELECT row: {author: Y, title: T}
         FROM X.author Y, X.title T)
FROM biblio.book X
WHERE "Roux" in X.author
```




```
Answer =
{row: {author:"Roux",
       title: "Database…"},
 row: {author:"Combalusier",
       title: "Database…"},
}
```

(Query has typo in the book ☹)

Formal Semantics

- Given query Q =

```
SELECT E[X1, …, Xn]
FROM   F
WHERE C
```

  and database DB
- Answer(Q,DB) is defined in two steps:
  - Step 1: compute all *bindings*:
    - C_{ij} are node oids or atomic values
    - Must satisfy paths in F
    - Must satisfy conditions in C

X_1	X_2	…	X_n
…	…	…	…
C_{i1}	C_{i2}	…	C_{in}
…	…	…	…

  - Step 2: answer is $E[C_{11}, …, C_{1n}] \cup … \cup E[C_{m1}, …, C_{mn}]$

Formal Semantics

- When E has nested subqueries, apply semantics recursively
- Note: so far we have dealt with an unordered model
  - What do we need to do for order ?
- Complexity: PTIME in |DB| (not in |Q|).

Lorel

- Minor syntactic differences in regular path expressions (% instead of _, # instead of _*)
- *Common path* convention:

```
SELECT biblio.book.author
FROM biblio.book
WHERE biblio.book.year = 1999
```

  becomes:

```
SELECT X.author
FROM biblio.book X
WHERE X.year = 1999
```

Lorel

- Existential variables:

```
SELECT biblio.book.year
FROM biblio.book
WHERE biblio.book.author = "Roux"
```

  - What happens with books having multiple authors ? Author is existentially quantified:

```
SELECT  X.year
FROM biblio.book X, X.author Y
WHERE Y =  "Roux"
```

Lorel

- Path variables. @P in:

```
SELECT P
FROM biblio.# @P X
```

  - What happens on graphs with cycles ?
- Constructing new results
  - Several default rules
- Casting between datatypes
  - Very useful in practice

UnQL

Patterns:

```
SELECT row: X
WHERE {biblio.book: {author "Roux",
                     title X}} in DB,
```

Equivalent to:

```
SELECT row: X
FROM   biblio.book Y,
       Y.author Z,
       Y.title X
WHERE Z="Roux"
```

UnQL

Label variables:

- "find all publication types and their titles where Roux is an author"

```
SELECT row: {type: L, title : X}
WHERE {biblio.L: {author "Roux",
                  title X}} in DB,
```

UnQL

Unrestricted use of label variables creates problems:

```
SELECT row: {type: L, title : Y}
WHERE {biblio.(book|L).title X} in DB,
```

```
SELECT row: {type: L, title : Y}
WHERE {biblio.(L)*.title X} in DB,
```

UnQL

In UnQL regular path expressions cannot contain label variables:

Pat ::= Var | Const | $\{L_1:Pat_1, \ldots, L_n:Pat_n\}$
L ::= RegularPathExpression | LabelVariable