Beyond relations: complex structure and XML

A history of databases in No-tation
1970: NoSQL = We have no SQL
1980: NoSQL = Know SQL
2000: NoSQL = No SQL!
2005: NoSQL = Not only SQL
2013: NoSQL = No, SQL!
A complex structure (nested relation)

Cinema:
set ( tuple ( Director: char, 
Movies: set ( tuple 
( Title: char,  Actors: set (char)))))

Other structures: list, bag

Example query:
find all movies whose director is not an actor in the movie

select m.Title
from f in Cinema, m in f.Movies
where f.Director not in m.Actors

Another example query:
find all directors that have not acted in any Hitchcock movie

select f.Director
from f in Cinema
where f.Director not in flatten
(select m.Actors
from g in Cinema, m in g.Movies
where g.Director = "Hitchcock")

flatten operator:
constructs the union of sets of actors

there are more operators to deal with complex structures
Another kind of nested data: XML

- W3C standard to complement HTML
- origins: structured text SGML
- motivation:
  -- HTML describes presentation
  -- XML describes content

From HTML to XML

Bibliography

*Foundations of Databases*, Abiteboul, Hull, Yianu
Addison Wesley, 1995

*Data on the Web*, Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999

HTML describes the presentation
**HTML**

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
    Abiteboul, Hull, Vianu
    <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
    Abiteboul, Buneman, Suciu
    <br> Morgan Kaufmann, 1999
```
Basic schema: Document Type Definition (DTD)

- root: paper
- paper → section*
- section → intro. section*.conclusions
- paper
  - section
    - intro section conc
    - intro section conc
    - intro conc
  - section
    - intro conc
    - intro conc
- UsedCars
- NewCars
- ad
  - model Honda
  - year 96
  - model Acura
Query languages for XML

Standard: XQuery
integrates previous paradigms

• One paradigm *(also XML-QL, Lorel, XMAS)*
  – extract bindings for variables using patterns
  – construct answer from bindings
• Another paradigm *(also XSLT)*
  – structural recursion

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

pattern:

```
p   q
X  Y
r
Z
s
T
```
p,q,r,s: regular path expressions
Basic form of XML-QL query

**WHERE** `pattern(X,Y,..)`

**CONSTRUCT** `answer(X,Y,..)`

```
\[ \text{pattern:} \quad \text{root} \quad \begin{array}{c}
p \\
X \\
pq \\
Y \\
pqrst \quad p,q,r,s: \text{regular path expressions} \\
Z=5 \\
T \\
n \end{array} \]
```
Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

answer:

root

f(X), g(X,Y)
constructor functions

---

Basic form of XML-QL query

WHERE pattern(X,Y,…)

CONSTRUCT answer(X,Y,…)

answer:

root

f(X), g(X,Y)
constructor functions
Basic form of XML-QL query

**WHERE** \( \text{pattern}(X,Y,\ldots) \)

**CONSTRUCT** \( \text{answer}(X,Y,\ldots) \)

**answer:**

\[ f(X_1) \ldots f(X_i) \ldots f(X_n) \]

constructor functions
Basic form of XML-QL query

**WHERE** pattern(X,Y,…)

**CONSTRUCT** answer(X,Y,…)

Example: art exhibits

DTD:

```
root                   → exhibit*
exhibit                → title, museum*, review*
title                  → artist*
museum                 → address, dates_ *
```

WHERE

```
root
    \|-- exhibit
        \|-- title
            \|-- museum
                \|-- artist
                    \|-- Vermeer
```

CONSTRUCT

```
Vermeer's exhibits
    \|-- title
        \|-- museum
            \|-- Q
```

```
answer:
      \|-- root
          \|-- f(X)
              \|-- g(X, Y_1)…g(X, Y_k)
```

```
constructor functions
      \|-- f(X)
```

```
exhibit
      \|-- title
```

```
museum
      \|-- address, dates_ *
```
Nested query $Q(X_2)$:

```
WHERE

root

exhibit

Y_1

title

review

X_2 Y_2

CONSTRUCT

Vermeer’s-reviews(X_2)

| review(X_2, Y_2) |
```

A Different Paradigm: Structural Recursion

Example: change all name tags occurring under person nodes to pname

- Notation: $a(t_1, t_2)$ denotes the tree

- Transformation defined by $f$:

\[
f(x(t_1, t_2)) = \begin{cases} 
    x = \text{person} & \text{then } x \ (g(t_1), g(t_2)) \\
    x = \text{name} & \text{then } x \ (f(t_1), f(t_2)) \\
    \text{else} & x \ (f(t_1), f(t_2)) 
\end{cases}
\]

\[
g(x(t_1, t_2)) = \begin{cases} 
    x = \text{name} & \text{then } \text{name} \ (g(t_1), g(t_2)) \\
    \text{else} & x \ (g(t_1), g(t_2)) 
\end{cases}
\]
Systems Issues

- XML servers
  storage: native vs. relational
- Indexing
  based on path expressions
- Query processing
  based on an XQuery algebra
- XML mediators

Integrated access to multiple data sources

Mediator

Data Source
Data Source

XML or relational db
Example: An XML Mediator

• source: relational database

<table>
<thead>
<tr>
<th>Store</th>
<th>SB</th>
<th>Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>sid</td>
<td>bid</td>
</tr>
<tr>
<td>name</td>
<td>bid</td>
<td>title</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

• virtual XML view:

```xml
<store>
  <name> n1 </name>
  <book> ... </book>
  <book> ... </book>
  ...
</store>

<store>
  <name> n2 </name>
  <book> ... </book>
  <book> ... </book>
  ...
</store>
```

Example: An XML Mediator

specify mediator declaratively (a view in SilkRoute):

```sql
from Store, SB, Book
where Store.sid=SB.sid and SB.bid=Book.bid
construct <store ID=f(Store.sid)>
  <name> Store.name </name>
  <book> Book.title </book>
</store>
```
Example: An XML Mediator

users ask XML-QL queries:

*Find the stores selling “The Calculus”*

```xml
where <store> <name> $n </name>
    <book> The Calculus </book>
<store>
construct <result> $n </result>
```

Example: An XML Mediator

• system *composes* query with view:

```sql
from Store, SB, Book
where Store.sid=SB.sid and
    SB.bid=Book.bid and
    Book.title=“The Calculus”
construct <result> Store.name </result>
```

• next step: *optimization* of resulting query