Lecture 5: Relational Algebra and XML

Monday, April 26th, 2004

1

Course Agenda

- Today, XML and relational algebra
- Next two weeks: the internals of DBMS.
  - Covered in gory detail in the book, but stay tuned for reading assignments.
- May 20th (not 17th!): Phil Bernstein on meta-data management.
- May 24th: data integration.
- May 27th: final exam.

2

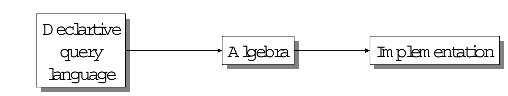
Agenda

- Relational algebra
- XML:
  - What is it and why do we care?
  - Data model
  - Query language: XPath
  - Real query language: XQuery.
  - General ruminations about XML.

3

Relational Algebra

- Formalism for creating new relations from existing ones
- Its place in the big picture:



SQL, relational calculus

Relational algebra
Relational bag algebra

4

Relational Algebra

- Five operators:
  - Union: ˘
  - Difference: -
  - Selection: s
  - Projection: P
  - Cartesian Product: ·
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural, equi-join, theta join, semi-join)
  - Renaming: r

5

1. Union and 2. Difference

- R1 ˘ R2
- Example:
  - ActiveEmployees ˘ RetiredEmployees
- R1 – R2
- Example:
  - AllEmployees – RetiredEmployees

6

What about Intersection ?

- It is a derived operator
- $R1 \cap R2 = R1 - (R1 - R2)$
- Also expressed as a join (will see later)
- Example
  - UnionizedEmployees $\cap$ RetiredEmployees

7

3. Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- Examples
  - $\sigma_{Salary > 40000}$ (Employee)
  - $\sigma_{name = \text{"Smith"}}$ (Employee)
- The condition c can be =, <, $\leq$, >, $\geq$, <>

8

Selection Example

Employee

SSN	Name	DepartmentID	Salary
999999999	John	1	30,000
777777777	Tony	1	32,000
888888888	Alice	2	45,000

Find all employees with salary more than $40,000.

$\sigma_{Salary > 40000}$ (Employee)

SSN	Name	DepartmentID	Salary
888888888	Alice	2	45,000

9

4. Projection

- Eliminates columns, then removes duplicates
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Example: project social-security number and names:
  - $\Pi_{SSN, Name}$ (Employee)
  - Output schema: Answer(SSN, Name)

10

Projection Example

Employee

SSN	Name	DepartmentID	Salary
999999999	John	1	30,000
777777777	Tony	1	32,000
888888888	Alice	2	45,000

$\Pi_{SSN, Name}$ (Employee)

SSN	Name
999999999	John
777777777	Tony
888888888	Alice

11

5. Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: $R1 \times R2$
- Example:
  - Employee $\times$ Dependents
- Very rare in practice; mainly used to express joins

12

Cartesian Product Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

EmployeeSSN	Dname
999999999	Emily
777777777	Joe

Employee x Dependents

Name	SSN	EmployeeSSN	Dname
John	999999999	999999999	Emily
John	999999999	777777777	Joe
Tony	777777777	999999999	Emily
Tony	777777777	777777777	Joe

13

Renaming

- Changes the schema, not the instance
- Notation: $r_{B1,\ldots,Bn}$ (R)
- Example:
  - $r_{LastName, SocSocNo}$ (Employee)
  - Output schema: Answer(LastName, SocSocNo)

14

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$r_{LastName, SocSocNo}$ (Employee)

LastName	SocSocNo
John	999999999
Tony	777777777

15

Natural Join

- Notation: R1 ⋈ R2
- Meaning: R1 ⋈ R2 = $P_A(s_C(R1 \cdot R2))$
- Where:
  - The selection s_C checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

16

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee ⋈ Dependents =

$P_{Name,SSN,Dname}(s_{SSN=SSN2}(\text{Employee x } r_{SSN2,Dname}(\text{Dependents})))$

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

17

Natural Join

- R =

A	B
X	Y
X	Z
Y	Z
Z	V

S =

B	C
Z	U
V	W
Z	V

- R ⋈ S =

A	B	C
X	Z	U
X	Z	V
Y	Z	U
Y	Z	V
Z	V	W

18

Natural Join

- Given the schemas R(A,B,C,D), S(A,C,E), what is the schema of $R \bowtie S$?
- Given R(A,B,C), S(D,E), what is $R \bowtie S$?
- Given R(A,B), S(A,B), what is $R \bowtie S$?

19

Theta Join

- A join that involves a predicate
- $R1 \bowtie_{q} R2 = s_{q} (R1 \cdot R2)$
- Here q can be any condition

20

Eq-join

- A theta join where q is an equality
- $R1 \bowtie_{A=B} R2 = s_{A=B} (R1 \cdot R2)$
- Example:
  - Employee $\bowtie_{SSN=SSN}$ Dependents
- Most useful join in practice

21

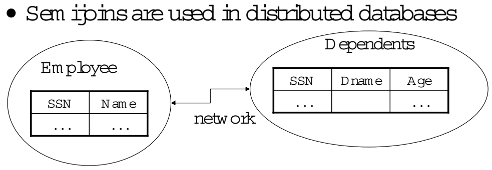
Semijoin

- $R \ltimes S = P_{A1,\ldots,An} (R \bowtie S)$
- Where $A_1, \ldots, A_n$ are the attributes in R
- Example:
  - Employee $\ltimes$ Dependents

22

Semijoins in Distributed Databases

- Semijoins are used in distributed databases



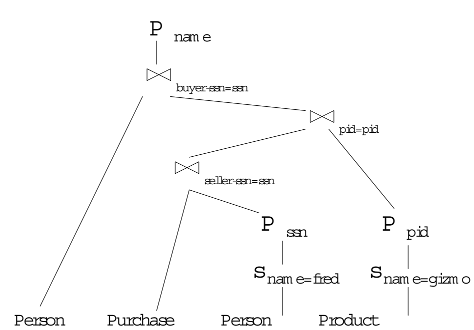
Employee $\bowtie_{ssn=ssn}$ ($s_{age>71}$ (Dependents))

R = Employee $\ltimes$ T

T = P_{SSN} $s_{age>71}$ (Dependents)

Answer = R $\bowtie$ Dependents

Complex RA Expressions



24

Operations on Bags

A bag = a set with repeated elements

All operations need to be defined carefully on bags

- {a,b,b,c} ˆ {a,b,b,b,e,f,f}={a,a,b,b,b,b,b,c,e,f,f}
- {a,b,b,b,c,c} – {b,c,c,c,d} = {a,b,b,d}
- s_C(R): preserve the number of occurrences
- P_A(R): no duplicate elimination
- Cartesian product, join: no duplicate elimination

Important! Relational Engines work on bags, not sets!

Reading assignment: 5.3 – 5.4

25

Finally: RA has Limitations!

- How do we compute "transitive closure"?

Name1	Name2	Relationship
Fred	Mary	Father
Mary	Joe	Cousin
Mary	Bill	Spouse
Nancy	Lou	Sister

- Find all direct and indirect relatives of Fred

26

XML

27

XML

- eXtensible Markup Language
- XML 1.0 – a recommendation from W3C, 1998
- Roots: SGML (a very nasty language).
- After the roots: a format for sharing data

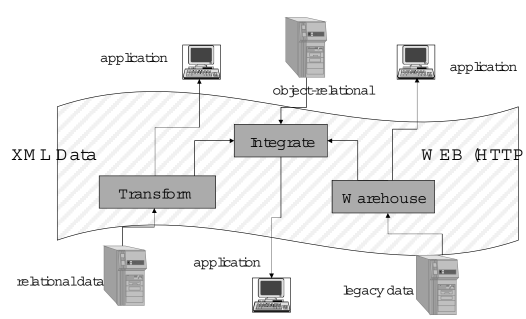
28

Why XML is of Interest to Us

- XML is just syntax for data
  - Note: we have no syntax for relational data
  - But XML is not relational: semistructured
- This is exciting because:
  - Can translate any data to XML
  - Can ship XML over the Web (HTTP)
  - Can input XML into any application
  - Thus: data sharing and exchange on the Web

29

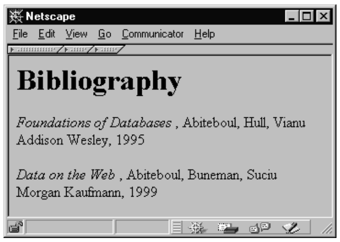
XML Data Sharing and Exchange



Specific data management tasks

30

From HTML to XML



HTML describes the presentation

31

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
      Abiteboul, Hull, Vianu
      <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
      Abiteoul, Buneman, Suciu
      <br> Morgan Kaufmann, 1999
```

32

XML

```
<bibliography>
    <book>   <title> Foundations… </title>
             <author> Abiteboul </author>
             <author> Hull </author>
             <author> Vianu </author>
             <publisher> Addison Wesley </publisher>
             <year> 1995 </year>
    </book>
    …
</bibliography>
```

XML describes the content

33

Web Services

- A new paradigm for creating distributed applications?
- Systems communicate via messages, contracts.
- Example: order processing system.
- MS .NET, J2EE – some of the platforms
- XML – a part of the story; the data format.

34

XML Terminology

- tags: book, title, author, …
- start tag: <book>, end tag: </book>
- elements: <book>… <book>,<author>… </author>
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single root element

well formed XML document: if it has matching tags

35

More XML: Attributes

```
<book price = "55" currency = "USD">
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
  …
  <year> 1995 </year>
</book>
```

attributes are alternative ways to represent data

36

More XML: Oids and References

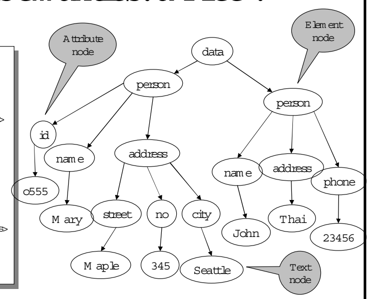
```
<person id="o555"> <name> Jane </name> </person>

<person id="o456"> <name> Mary </name>
                <children idref="o123 o555"/>
</person>

<person id="o123" mother="o456"><name>John</name>
</person>
```

oids and references in XML are just syntax

37

XML Semantics: a Tree !

```
<data>
    <person id="o555" >
        <name> Mary </name>
        <address>
            <street> Maple </street>
            <no> 345 </no>
            <city> Seattle </city>
        </address>
    </person>
    <person>
        <name> John </name>
        <address> Thailand </address>
        <phone> 23456 </phone>
    </person>
</data>
```



Order matters !!!

38

XML Data

- XML is self-describing
- Schema elements become part of the data
  - Reational schema: persons(name,phone)
  - In XML <persons>, <name>, <phone> are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

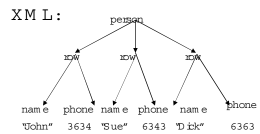
39

Relational Data as XML

person

name	phone
John	3634
Sue	6343
Dick	6363

XML:




```
<person>
    <row> <name>John</name>
            <phone> 3634</phone></row>
    <row> <name>Sue</name>
            <phone> 6343</phone>
    <row> <name>Dick</name>
            <phone> 6363</phone></row>
</person>
```

XML is Semi-structured Data

- Missing attributes:

```
<person> <name> John</name>
         <phone>1234</phone>
</person>

<person> <name>Joe</name>
</person>
```

no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

41

XML is Semi-structured Data

- Repeated attributes

```
<person> <name> Mary</name>
         <phone>2345</phone>
         <phone>3456</phone>
</person>
```

two phones !

- Impossible in tables:

name	phone	
Mary	2345	3456

???

42

XML is Semi-structured Data

- Attributes with different types in different objects

```
<person> <name> <first> John </first>
                <last> Smith </last>
         </name>
         <phone>1234</phone>
</person>
```

structured name !

- Nested collections (no 1NF)
- Heterogeneous collections:
  - <db> contains both <book>s and <publisher>s

43

Document Type Definitions DTD

- part of the original XML specification
- an XML document may have a DTD
- XML document:
  well-formed = if tags are correctly closed
  Valid = if it has a DTD and conforms to it
- validation is useful in data exchange

44

Very Simple DTD

```
<!DOCTYPE company [
 <!ELEMENT company  ((person|product)*)>
 <!ELEMENT person  (ssn,name,office,phone?)>
 <!ELEMENT ssn     (#PCDATA)>
 <!ELEMENT name    (#PCDATA)>
 <!ELEMENT office  (#PCDATA)>
 <!ELEMENT phone   (#PCDATA)>
 <!ELEMENT product (pid,name,description?)>
 <!ELEMENT pid   (#PCDATA)>
 <!ELEMENT description   (#PCDATA)>
]>
```

45

Very Simple DTD

Example of valid XML document:

```
<company>
  <person> <ssn> 123456789 </ssn>
           <name> John </name>
           <office> B432 </office>
           <phone> 1234 </phone>
  </person>
  <person> <ssn> 987654321 </ssn>
           <name> Jim </name>
           <office> B123 </office>
  </person>
  <product> ... </product>
  ...
</company>
```

46

DTD: The Content Model

```
<!ELEMENT tag (CONTENT)>
```

content model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*

47

DTD: Regular Expressions

DTD

XML

sequence

```
<!ELEMENT name
          (firstName, lastName))
```

```
<name>
  <firstName> .....</firstName>
  <lastName> .....</lastName>
</name>
```

optional

```
<!ELEMENT name  (firstName?, lastName))
```

Kleene star

```
<!ELEMENT person   (name, phone*))
```

```
<person>
  <name> .....</name>
  <phone> .....</phone>
  <phone> .....</phone>
  <phone> .....</phone>
  ......
</person>
```

alternation

```
<!ELEMENT person   (name, (phone|email)))
```

48

Querying XML Data

- XPath = simple navigation through the tree
- XQuery = the SQL of XML
- XSLT = recursive traversal
  - will not discuss in class

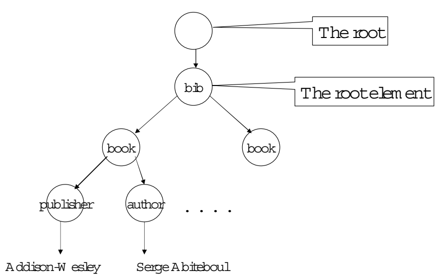
49

Sample Data for Queries

```
<bib>
  <book>  <publisher> Addison-Wesley </publisher>
          <author> Serge Abiteboul </author>
          <author> <first-name> Rick </first-name>
                   <last-name> Hull </last-name>
          </author>
          <author> Victor Vianu </author>
          <title> Foundations of Databases </title>
          <year> 1995 </year>
  </book>
  <book price="55">
          <publisher> Freeman </publisher>
          <author> Jeffrey D. Ullman </author>
          <title> Principles of Database and Knowledge Base Systems </title>
          <year> 1998 </year>
  </book>
</bib>
```

50

Data Model for XPath



51

XPath: Simple Expressions

/bib/book/year

Result: <year> 1995 </year>
<year> 1998 </year>

/bib/paper/year

Result: empty (there were no papers)

52

XPath: Restricted Kleene Closure

//author

Result: <author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>

/bib//first-name Rick </first-name>

53

Xpath: Text Nodes

/bib/book/author/text()

Result: Serge Abiteboul
Jeffrey D. Ullman

Rick Hull doesn't appear because he has firstname, lastname

Functions in XPath:
- text() = matches the text value
- node() = matches any node (= * or @* or text())
- name() = returns the name of the current tag

54

Xpath: Wildcard

//author/*

Result: <first-name> Rick </first-name>
<last-name> Hull </last-name>

* Matches any element

55

Xpath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price is has to be an attribute

56

Xpath: Predicates

/bib/book/author[firstname]

Result: <author> <first-name> Rick </first-name>
<last-name> Hull </last-name>
</author>

57

Xpath: More Predicates

/bib/book/author[firstname][address[//zip][city]]/lastname

Result: <lastname> … </lastname>
<lastname> … </lastname>

58

Xpath: More Predicates

/bib/book[@price < “60”]

/bib/book[author/@age < “25”]

/bib/book[author/text()]

59

Xpath: Summary

bib	matches a bib element
*	matches any element
/	matches the root element
/bib	matches a bib element under root
bib/paper	matches a paper in bib
bib//paper	matches a paper in bib, at any depth
//paper	matches a paper at any depth
paper|book	matches a paper or a book
@price	matches a price attribute

bib/book/@price matches price attribute in book, in bib

bib/book/[@price< “55”]/author/lastname matches…

60

Comments on XPath?

- What's good about it?
- What can't it do that you want it to do?
- How does it compare, say, to SQL?

XQuery

- Based on Quilt, which is based on XML-QL
- Uses XPath to express more complex queries

FLWR ("Flower") Expressions

```
FOR ...
LET ...
WHERE ...
RETURN ...
```

XQuery

Find all book titles published after 1995:

```
FOR $x IN document("bib.xml")/bib/book
WHERE $x/year > 1995
RETURN { $x/title }
```

Result:

```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```


XQuery

Find book titles by the coauthors of "Database Theory":

```
FOR $x IN bib/book[title/text() = "Database Theory"]/author
    $y IN bib/book[author/text() = $x/text()]/title
RETURN <answer> { $y/text() } </answer>
```

The answer will contain duplicates !

Result:

```
<answer> abc </answer>
<answer> def </answer>
<answer> ghi </answer>
```


XQuery

Same as before, but eliminate duplicates:

```
FOR $x IN bib/book[title/text() = "Database Theory"]/author
    $y IN distinct(bib/book[author/text() = $x/text()]/title)
RETURN <answer> { $y/text() } </answer>
```

distinct = a function that eliminates duplicates

Result:

```
<answer> abc </answer>
<answer> def </answer>
<answer> ghi </answer>
```

XQuery: Nesting

For each author of a book by Morgan Kaufmann, list all books she published:

```
FOR $a IN distinct(document("bib.xml")
            /bib/book[publisher="Morgan Kaufmann"]/author)
RETURN <result>
         { $a,
           FOR $t IN /bib/book[author=$a]/title
           RETURN $t
         }
       </result>
```

67

XQuery

Result:

```
<result>
    <author>Jones</author>
    <title> abc </title>
    <title> def </title>
</result>
<result>
    <author> Smith </author>
    <title> ghi </title>
</result>
```

68

XQuery

- FOR $x in expr -- binds $x to each value in the list expr
- LET $x = expr -- binds $x to the entire list expr
  - Useful for common subexpressions and for aggregations

69

XQuery

```
<big_publishers>
    FOR $p IN distinct(document("bib.xml")//publisher)
    LET $b := document("bib.xml")/book[publisher = $p]
    WHERE count($b) > 100
    RETURN { $p }
</big_publishers>
```

count = a (aggregate) function that returns the number of elms

70

XQuery

Find books whose price is larger than average:

```
LET $a=avg(document("bib.xml")/bib/book/price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/price > $a
RETURN { $b }
```

Let's try to write this in SQL...

71

XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR

FOR/LET Clauses

↓ List of tuples

WHERE Clause

↓ List of tuples

RETURN Clause

Instance of Xquery data model

72

FOR v.s. LET

FOR

- Binds node variables iteration

LET

- Binds collection variables one value

73

FOR v.s. LET

```
FOR $x IN document("bib.xml")/bib/book
RETURN <result> { $x } </result>
```

Returns:

```
<result> <book>...</book></result>
<result> <book>...</book></result>
<result> <book>...</book></result>
...
```

```
LET $x IN document("bib.xml")/bib/book
RETURN <result> { $x } </result>
```

Returns:

```
<result> <book>...</book>
         <book>...</book>
         <book>...</book>
         ...
</result>
```

74

Collections in XQuery

- Ordered and unordered collections
  - /bib/book/author = an ordered collection
  - Distinct(/bib/book/author) = an unordered collection
- LET $a = /bib/book $a is a collection
- $b/author a collection (several authors...)

```
RETURN <result> { $b/author } </result>
```

Returns:

```
<result> <author>...</author>
         <author>...</author>
         <author>...</author>
         ...
</result>
```

75

Collections in XQuery

What about collections in expressions ?

- $b/price list of n prices
- $b/price * 0.7 list of n numbers
- $b/price * $b/quantity list of n x m numbers ??
- $b/price * ($b/quant1 + $b/quant2) „ $b/price * $b/quant1 + $b/price * $b/quant2 !!

76

Sorting in XQuery

```
<publisher_list>
  FOR $p IN distinct(document("bib.xml")//publisher)
  RETURN <publisher> <name> { $p/text() } </name> ,
            FOR $b IN document("bib.xml")//book[publisher = $p]
            RETURN <book>
                      { $b/title ,
                        $b/price
                      }
                   </book> SORTBY(price DESCENDING)
         </publisher> SORTBY(name)
</publisher_list>
```

77

If-Then-Else

```
FOR $h IN //holding
RETURN <holding>
          { $h/title,
            IF $h/@type = "Journal"
               THEN $h/editor
            ELSE $h/author
          }
       </holding> SORTBY (title)
```

78

Existential Quantifiers

```
FOR $b IN //book
WHERE SOME $p IN $b//para SATISFIES
            contains($p, "sailing")
            AND contains($p, "windsurfing")
RETURN { $b/title }
```

79

Universal Quantifiers

```
FOR $b IN //book
WHERE EVERY $p IN $b//para SATISFIES
            contains($p, "sailing")
RETURN { $b/title }
```

80

Other Stuff in XQuery

- BEFORE and AFTER
  - for dealing with order in the input
- FILTER
  - deletes some edges in the result tree
- Recursive functions
  - Currently: arbitrary recursion
  - Perhaps more restrictions in the future ?

81

Final Comments on XML

- How are we going to process XML efficiently?
  - Special purpose XML engines, or
  - Add functionality to relational engines?
- Need to manage XML streams.
- Here, data management is much closer to other programming tasks.

82