Lecture 5: Relational Algebra and XML

Monday, April 26th, 2004

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.

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.

Relational Algebra

- Formalism for creating new relations from existing ones
- Its place in the big picture:
  - Relational calculus
  - SQL, relational algebra
  - Relational bag algebra

Relational Algebra

- Five operators:
  - Union: ∪
  - Difference: −
  - Selection: s
  - Projection: P
  - Cartesian Product: ×

- Derived or auxiliary operators:
  - Intersection, compliment
  - Joins (natural, equi-join, theta join, semi-join)
  - Rename: r

1. Union and 2. Difference

- R1 ∪ R2
- Example:
  - ActiveEmployees ∪ RetiredEmployees

- R1 − R2
- Example:
  - AllEmployees − RetiredEmployees
What about Intersection?
- It is a derived operator
- \( R_1 \cap R_2 = R_1 - (R_1 - R_2) \)
- Also expressed as a join (we'll see later)
- Example:
  - \( \text{UnionizedEmployees} \cap \text{RetiredEmployees} \)

3. Selection
- Returns all tuples which satisfy a condition
- Notation: \( s_c(R) \)
- Examples:
  - \( s_{\text{Salary} > 40000} \text{(Employee)} \)
  - \( s_{\text{Name} = \text{"Smith"}} \text{(Employee)} \)
- The condition \( c \) can be: \( =, <, >, \neq, <, > \)

Selection Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Department</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>John</td>
<td>1</td>
<td>30,000</td>
</tr>
<tr>
<td>777777777</td>
<td>Tony</td>
<td>1</td>
<td>32,000</td>
</tr>
<tr>
<td>888888888</td>
<td>Alice</td>
<td>2</td>
<td>45,000</td>
</tr>
</tbody>
</table>

Find all employees with salary more than $40,000.
\( s_{\text{Salary} > 40000} \text{(Employee)} \)

4. Projection
- Eliminates columns, then removes duplicates
- Notation: \( P_{A_1, A_2, \ldots, A_n}(R) \)
- Example: project social-security number and names:
  - \( P_{\text{SSN}, \text{Name}} \text{(Employee)} \)
  - Output schema: \( \text{Answer( SSN, Name )} \)

Projection Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>John</td>
</tr>
<tr>
<td>777777777</td>
<td>Tony</td>
</tr>
<tr>
<td>888888888</td>
<td>Alice</td>
</tr>
</tbody>
</table>

\( P_{\text{SSN}, \text{Name}}( \text{Employee} ) \)

5. Cartesian Product
- Each tuple in \( R_1 \) with each tuple in \( R_2 \)
- Notation: \( R_1 \times R_2 \)
- Example:
  - \( \text{Employee} \times \text{Dependents} \)
- Very rare in practice; mainly used to express joins
### Cartesian Product Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

### Dependents

<table>
<thead>
<tr>
<th>Employee SSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

### Employee x Dependents

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Employee SSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>John</td>
<td>999999999</td>
<td>777777777</td>
<td>Joe</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

### Renaming

- Changes the schema, not the instance
- Notation: \( r_{B_1, \ldots, B_n}(R) \)
- Example:
  - \( r_{\text{LastName}, \text{SocSocNo}}(\text{Employee}) \)
  - Output schema: \( \text{Answer}((\text{LastName}, \text{SocSocNo})) \)

### Natural Join

- Notation: \( R_1 \bowtie R_2 \)
- Meaning: \( R_1 \bowtie R_2 = P_{\alpha}(s_{\beta}(R_1 \cdot R_2)) \)
- Where:
  - The selection \( s_{\beta} \) checks equality of all common attributes
  - The projection eliminates the duplicate common attributes

### Natural Join Example

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

\[ \text{Employee} \bowtie \text{Dependents} = P_{\text{Name}, \text{SSN}, \text{Dname}}(s_{\text{SSN}=\text{SSN}}(\text{Employee} \times r_{\text{SSN}=\text{SSN}}(\text{Dependents}))) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>SSN</th>
<th>Dname</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>999999999</td>
<td>Emily</td>
</tr>
<tr>
<td>Tony</td>
<td>777777777</td>
<td>Joe</td>
</tr>
</tbody>
</table>

### Natural Join

<table>
<thead>
<tr>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
</tbody>
</table>

\[ R \bowtie S = \begin{bmatrix} A & B & C \\ X & 0 & 0 \\ X & Z & V \\ Y & Z & U \\ Y & V & Y \end{bmatrix} \]

### Natural Join

<table>
<thead>
<tr>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
</tbody>
</table>

\[ R \bowtie S = \begin{bmatrix} A & B & C \\ X & Z & U \\ X & Z & V \\ Y & Z & U \\ Y & V & Y \end{bmatrix} \]
**Natural Join**

- Given the schema as \( R(A, B, C, D), S(A, C, E) \), 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 \( R \bowtie S \)?

**Theta Join**

- A join that involves a predicate
- \( R1 \bowtie q R2 = s_q (R1 \cdot R2) \)
- Here \( q \) can be any condition

**Eq-join**

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

**Semijoin**

- \( R \bowtie S = P_{A_1,...,A_n} (R \bowtie S) \)
- Where \( A_1,...,A_n \) are the attributes in \( R \)
- Example:
  - Employee \( \bowtie \)Dependents
- Most useful join in practice

**Semijoins in Distributed Databases**

- Semijoins are used in distributed databases
- Example:
  - Employee \( \bowtie \)Dependents
  - \( \)Employee \( \bowtie \)Dependents

**Complex RA Expressions**

- Person \( \bowtie \)Purchase \( \bowtie \)Person \( \bowtie \)Product
- \( s_{\text{name}} \bowtie \text{ssn} \bowtie \text{name} \bowtie \text{pid} \)
- \( s_{\text{name}} \bowtie \text{ssn} \bowtie \text{name} \bowtie \text{pid} \)
- \( s_{\text{name}} \bowtie \text{ssn} \bowtie \text{name} \bowtie \text{pid} \)
Operations on Bags

A bag = a set with repeated elements

All operations need to be defined carefully on bags

- \( \{a, b, b, c\} \)
- \( \{a, b, b, b, e, f, f\} = \{a, a, b, b, b, b, b, c, e, f, f\} \)
- \( \{a, b, b, b, c, c\} \) – \( \{b, c, c, c, d\} = \{a, b, b, d\} \)
- \( \pi_{\alpha}(R) \): preserve the number of occurrences
- \( \pi_{\beta}(R) \): no duplicate elimination
- Cartesian product, join: no duplicate elimination

Important! Relational Engines work on bags, not sets!

Reading assignment: 5.3 – 5.4

Finally: RA has Limitations!

- How do we compute “transitive closure”?

<table>
<thead>
<tr>
<th>Name1</th>
<th>Name2</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fred</td>
<td>May</td>
<td>Parent</td>
</tr>
<tr>
<td>May</td>
<td>Joe</td>
<td>Cousin</td>
</tr>
<tr>
<td>May</td>
<td>Bill</td>
<td>Spouse</td>
</tr>
<tr>
<td>May</td>
<td>Lou</td>
<td>Sister</td>
</tr>
</tbody>
</table>

Find all direct and indirect relatives of Fred

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: semi-structured
- 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

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

XML Data Sharing and Exchange
From HTML to XML

HTML describes the presentation

From HTML to XML

HTML

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

XML describes the content

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

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.

```
• XML Terminology
• tags: book, title, author;...
• start tag: <book>, end tag: </book>
• elements: <book>; <author>... </author>
• elements are nested
• an empty element: <red/> abbrev. <red/>
• an XML document: single root element:

• well-formed XML document: if it has matching tags
```

```
• 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
XML Data
- XML is self-describing
- Schema elements become part of the data
  - Relational 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 = semi-structured data

XML is Semi-structured Data
- Missing attributes:
  - Could represent in a table with nulls
  - Repeated attributes
  - Impossible in tables

Relational Data as XML
- person
  <row> <name>John</name> <phone>3634</phone></row>
  <row> <name>Sue</name> <phone>6343</phone></row>
  <row> <name>Dick</name> <phone>6363</phone></row>

XML is Semi-structured Data
- Missing attributes, Repeated attributes
- Impossible in tables
XML is Semi-structured Data

- Attributes with different types in different objects
- Nested collections (no 1NF)
- Heterogeneous collections:
  - <db> contains both <book>s and <publisher>s

Structured name!

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

Very Simple DTD

Example of valid XML document:

```
<html>
  <body>
    <h1>Example Document</h1>
    <p>This is a simple paragraph.</p>
  </body>
</html>
```

DTD: The Content Model

```
<!ELEMENT person (ssn, name, office, phone?)>
```

DTD: Regular Expressions

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

Optional

```
<!ELEMENT product (pid, name, description?)>
```

Kleene star

```
<!ELEMENT person (ssn), (phone|email))>
```

Alternation

```
<!ELEMENT person (ssn), (phone|email))>
```
Querying XML Data

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

Sample Data for Queries

```xml
<bib>
  <book>
    <publisher>Addison-Wesley</publisher>
    <author>Serge Abiteboul</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
  <book>
    <publisher>Freeman</publisher>
    <author>Jeffrey D. Ullman</author>
    <title>Principles of Database and Knowledge Base Systems</title>
    <year>1998</year>
  </book>
</bib>
```

Data Model for XPath

- The root element is `<bib>`.
- Child elements include `<book>`, `<author>`, `<title>`, and `<year>`.

XPath: Simple Expressions

- `//author` returns empty, as there were no papers.

XPath: Restricted Kleene Closure

- `//author` returns `<author>Serge Abiteboul</author>`, `<author>Rick Hull</author>`, and `<author>Jeffrey D. Ullman</author>`.
- `//first-name` returns empty, as 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
**Xpath: Wildcard**

```
//author/*
```

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

* Matches any element.

---

**Xpath: Attribute Nodes**

```
/bib/book/@price
```

Result: "$55"

@ price means that price has to be an attribute.

---

**Xpath: Predicates**

```
/bib/book/author[firstname]
```

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

---

**Xpath: More Predicates**

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

Result: `<last-name>` ... </last-name>  
        `<last-name>` ... </last-name>`

---

**Xpath: More Predicates**

```
/bib/book[@price < "$60"]
```

```
/bib/book/author[@age < "25"]
```

```
/bib/book/author/text()
```

**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.  
@ matches a price attribute.  
bib/book[@price="$55"]/@author[lastname]/matches any element.
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 distinct(bib/book[author/text() = $x/text()]/title)
RETURN <answer> { $y/text() } </answer>

Result:
<answer> abc </answer>
<answer> def </answer>
<answer> ghi </answer>

The answer will contain duplicates!

distinct = a function that eliminates duplicates

distinct: $x$ IN $L$ $y$ IN $L$ $x \neq y$
XQuery: Nesting

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

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

Result:

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

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

Summary:

• FOR-LET-WHERE-RETURN = FLWR

XQuery

```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>
```

XQuery

Find books whose price is larger than average:

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

Summary:

• FOR-LET-WHERE-RETURN = FLWR

XQuery

```
FOR-LET Clauses
```

List of tuples

```
WHERE Clause
```

List of tuples

```
RETURN Clause
```

List of tuples

```
Instance of XQuery data model
```

Let's try to write this in SQL...
FOR vs. LET

FOR
• Binds node variables
• Iteration

LET
• Binds collection variables
• One value

Collections in XQuery

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

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

Returns:
<result> <author>...</author>
<author>...</author>
<author>...</author>
...

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??
• $b/price$ * $b/quantity$ + $b/price$ * $b/quantity$ ??

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>
</publisher_list>

If-Then-Else

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

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

Universal Quantifiers

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

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 some restrictions in the future?

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.