CSE544: Lecture 7

XQuery, Relational Algebra

Monday, 4/22/02

XQuery

- Based on Quilt
  (which is based on XML-QL)
- http://www.w3.org/XML/Query
- XML Query data model
  - Similar to the XPath data model, more complete

FLWR (“Flower”) Expressions

FOR ... LET... 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

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>
```

distinct = a function that eliminates duplicates

XQuery

Result:

```
<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

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

XQuery

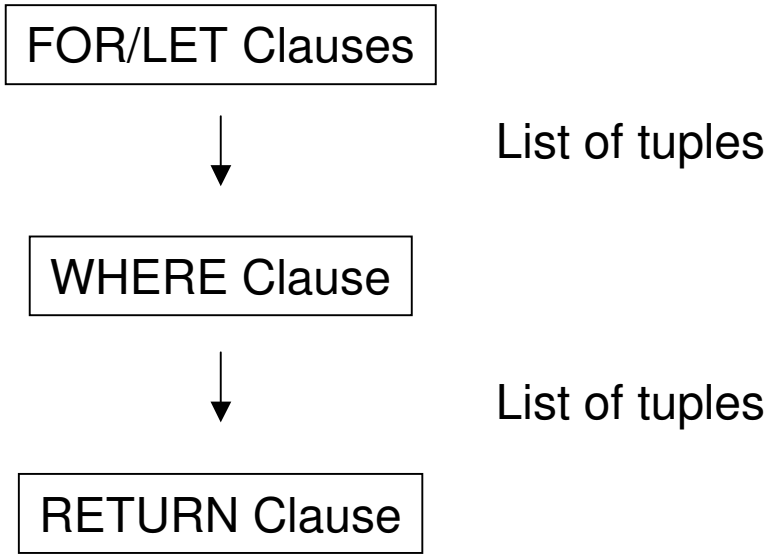
Find books whose price is larger than average:

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

XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR



FOR v.s. LET

FOR

- Binds *node variables* → iteration

LET

- Binds *collection variables* → one value

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>
```

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>
```

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 !!

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>
```

Sorting in XQuery

- Sorting arguments: refer to the name space of the RETURN clause, not the FOR clause

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 more restrictions in the future ?

Foundations of Database Systems

- Why is theory important ?

- Roadmap to database theory in CSE544
  - Relational algebra (today)
  - First order logic (a.k.a. relational calculus)
  - Conjunctive queries and datalog

Relational Algebra at a Glance

- Algebra on relations
  - Set algebra: e.g. the boolean algebra
  - Algebra on relations: e.g. Tarski's cylindrical algebra
- Five basic RA operators:
  - Union, difference (from the boolean algebra): $\cup$, -
  - Selection: σ
  - Projection: π
  - Cartesian Product: $\times$
- Derived operators: intersection, complement, joins
- Renaming: ρ

Union

- Union: all tuples in R1 or R2
- Notation: R1 $\cup$ R2
- R1, R2 must have the same schema
- R1 $\cup$ R2 has the same schema as R1, R2
- Example:
  - ActiveEmployees $\cup$ RetiredEmployees

Difference

- Difference: all tuples in R1 and not in R2
- Notation: R1 – R2
- R1, R2 must have the same schema
- R1 – R2 has the same schema as R1, R2
- Example
  - AllEmployees – RetiredEmployees

Intersection

- Difference: all tuples both in R1 and in R2
- Notation: R1 ∩ R2
- R1, R2 must have the same schema
- R1 ∩ R2 has the same schema as R1, R2
- Example
  - UnionizedEmployees ∩ RetiredEmployees
- Derived operation:
  - R1 ∩ R2 = R1 – (R1 – R2)

Selection

- Returns all tuples which satisfy a condition
- Notation: $\sigma_c(R)$
- *c* is a condition: =, <, >, and, or, not
- Output schema: same as input schema
- Find all employees with salary more than $40,000:
  - $\sigma_{Salary > 40000}$ (Employee)

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.

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

SSN	Name	DepartmentID	Salary
888888888	Alice	2	45,000

Projection

- Unary operation: returns certain columns
- Eliminates duplicate tuples !
- Notation: $\Pi_{A1,\ldots,An}(R)$
- Input schema R(B1,…,Bm)
- Condition: {A1, …, An} $\subseteq$ {B1, …, Bm}
- Output schema S(A1,…,An)
- Example: project social-security number and names:
  - $\Pi_{SSN,\ Name}$ (Employee)

Projection Example

Employee

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

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

SSN	Name
999999999	John
777777777	Tony
888888888	Alice

Cartesian Product

- Each tuple in R1 with each tuple in R2
- Notation: $R1 \times R2$
- Input schemas R1(A1,…,An), R2(B1,…,Bm)
- Condition: $\{A1,\ldots,An\} \cap \{B1,\ldots Bm\} = \Phi$
- Output schema is S(A1, …, An, B1, …, Bm)
- Notation: $R1 \times R2$
- Example: **Employee × Dependents**
- Very rare in practice; but joins are very often

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

Renaming

- Does not change the relational instance
- Changes the relational schema only
- Notation: $\rho_{B1,\ldots,Bn}(R)$
- Input schema: R(A1, …, An)
- Output schema: S(B1, …, Bn)
- Example:

$$\rho_{\text{LastName, SocSocNo}}(\text{Employee})$$

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$\rho_{LastName,\ SocSocNo}$ **(Employee)**

LastName	SocSocNo
John	999999999
Tony	777777777

Natural Join

- Notation: R1 $\bowtie$ R2
- Input Schema: R1(A1, …, An), R2(B1, …, Bm)
- Output Schema: S(C1,…,Cp)
  - Where {C1, …, Cp} = {A1, …, An} $\cup$ {B1, …, Bm}
- Meaning: combine all pairs of tuples in R1 and R2 that agree on the attributes:
  - {A1,…,An} $\cap$ {B1,…, Bm} (called the join attributes)
- Equivalent to a cross product followed by selection
- Example **Employee** $\bowtie$ **Dependents**

Natural Join Example

Employee

Name	SSN
John	999999999
Tony	777777777

Dependents

SSN	Dname
999999999	Emily
777777777	Joe

Employee $\bowtie$ Dependents =

$$\Pi_{\text{Name, SSN, Dname}}(\sigma_{\text{SSN=SSN2}}(\text{Employee} \times \rho_{\text{SSN2, Dname}}(\text{Dependents}))$$

Name	SSN	Dname
John	999999999	Emily
Tony	777777777	Joe

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

Natural Join

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

Theta Join

- A join that involves a predicate
- Notation: $R1 \bowtie_\theta R2$ where θ is a condition
- Input schemas: R1(A1,…,An), R2(B1,…,Bm)
- $\{A1,\ldots An\} \cap \{B1,\ldots,Bm\} = \phi$
- Output schema: S(A1,…,An,B1,…,Bm)
- Derived operator:

  $R1 \bowtie_\theta R2 = \sigma_\theta (R1 \times R2)$

Eq-join

- Most frequently used in practice:

$$R1 \bowtie_{A=B} R2$$

- Natural join is a particular case of eqjoin
- A lot of research on how to do it efficiently

Semijoin

- $R \ltimes S = \Pi_{A1,\ldots,An}(R \bowtie S)$
- Where the schemas are:
  - Input: R(A1,…An), S(B1,…,Bm)
  - Output: T(A1,…,An)

Semijoin

Applications in distributed databases:

- Product(<u>pid</u>, cid, pname, ...) at site 1
- Company(<u>cid</u>, cname, ...) at site 2
- Query: $\sigma_{price>1000}(Product) \ltimes_{cid=cid} Company$
- Compute as follows:

$T1 = \sigma_{price>1000}(Product)$	site 1
$T2 = \Pi_{cid}(T1)$	site 1
send T2 to site 2	(T2 smaller than T1)
$T3 = T2 \ltimes Company$	site 2 (semijoin)
send T3 to site 1	(T3 smaller than Company)
$Answer = T3 \ltimes T1$	site 1 (semijoin)

Relational Algebra Summary

- Five basic operators, many derived
- Combine operators in order to construct queries: relational algebra expressions, usually shown as trees