XML Query Languages: XQUERY

Contains slides made by
Naci Akkøk, Pål Halvorsen, Arthur M. Keller, Vera Goebel

Querying XML Data

- "The goal of the XML Query WG is to produce a data model for XML documents, a set of query operators on that data model, and a query language based on these query operators."
- XML query languages: XPATH, XPOINTER, and *XQUERY*
- XQUERY (see: http://www.w3.org/TR/xquery/):
  - is an emerging standard for querying XML documents
  - strongly influenced by OQL
  - XQUERY is a functional language in which a query is represented as an expression (opposed to OQL and SQL which are declarative)
  - XQUERY expressions can be nested
  - filters can strip out fields
  - grouping

XQUERY – I

- XQUERY provides a FLWR expression:
  - F – FOR: associates with variables, creating an ordered sequence of tuples drawn form the cartesian product of the variables. It iterates through a sequence of individual nodes out of the selected collection, in order, one at a time
  - L – LET: binds a variable directly to an entire expression – to the set of nodes in the selected collection
  - W – WHERE: predicates used on bound variables, used as a filter for the tuples generated by the FOR and LET clauses
  - R – RETURN: contains an expression that is used to construct the result from the whole FLWR expression. Invoked for every tuple generated by the FOR and LET clauses, but after eliminating any tuples in the WHERE clause

XQUERY:
FLWR expressions – I

- FLWR expressions:
  `(FORexpr | LETexpr)+` `WHEREclause?` `RETURNexpr`

  - `FORexpr`: `FOR` variable `IN` expression (, variable `IN` expression)*
  - `LETexpr`: `LET` variable `:=` expression (, variable `:=` expression)*
  - `WHEREexpr`: `WHERE` expression
  - `RETURNexpr`: `RETURN` expression

NOTE 1:
`FOR` and / or `LET` appear one or more times

NOTE 2:
`WHERE` clauses are optional

NOTE 3:
a `RETURN` clause is always present

XQUERY:
FLWR Examples – I

- Example 1:

```
LET $a := (1, 2, 3)
RETURN <out>{$a}</out>
```

Output:

```
<out>1 2 3</out>
```

- Example 2:

```
FOR $a IN (1, 2, 3)
RETURN <out>{$a}</out>
```

Output:

```
<out>1</out>
<out>2</out>
<out>3</out>
```

NOTE 1a:
the variable `$a` is bound to the expression `(1, 2, 3)`. LET clause generates one tuple containing the variable binding of `$a`

NOTE 1b:
one might add tags in the output, i.e., in the `RETURN` clause

NOTE 2:
the variable `$a` is associated with the expression `(1, 2, 3)` from which the variable bindings of `$a` will be drawn, i.e., `$a` will be processed in such a way that the value of `$a` will be bound for each element in the expression – in this case three times

XQUERY:
FLWR Examples – II

- Example 3:

```
FOR   $a IN (1, 2),
      $b IN (a, b)
RETURN      <out>
               <a>{$a}</a> <b>{$b}</b>
            </out>
```

NOTE 3a:
we may have multiple FOR clauses, each variable associated with an expression

Output:

```
<out>
    <a>1</a> <b>a</b>
    <a>1</a> <b>b</b>
    <a>2</a> <b>a</b>
    <a>2</a> <b>b</b>
</out>
```

NOTE 3b:
the tuples are drawn from the cartesian product of the sequence returned in FOR, i.e., cartesian product of $a and $b.

NOTE 3c:
the order of the tuples are the order of which they were formed – from left to right and variable $a before $b

XQUERY:
FLWR expressions – II

- `FOR` and `LET` clauses operate on sets
- Sets of elements can be described by paths, consisting of:
  1. URL or file name, e.g.,
     `$ba IN document("bars.xml")` – "bars.xml" contain data for `$ba`
  2. elements forming a path in the semi-structured data graph, e.g.,
     `//BAR/NAME` – start at any `BAR` node and go to a `NAME` child
  3. ending condition of the form the path
     `[`<sub-elements conditions, @attributes, and values>`]`, e.g.,
     `//BAR/BEER[NAME = "Bud"]` – beer elements in a bar where there is a beer named "Bud"
     `//BAR[@TYPE = "Sports"]` – bar elements whose attribute named type has value "Sports"
  4. ....

XQUERY:
BBS Example – I

```
<?XML VERSION = "1.0" STANDALONE = "no"?>
<!DOCTYPE Bars SYSTEM "bar.dtd">
<BARS>
    <BAR type = "sports">
            <NAME>Joe's</NAME>
            <BEER><NAME>Bud</NAME>
                    <PRICE>2.50</PRICE></BEER>
            <BEER><NAME>Miller</NAME>
                    <PRICE>3.00</PRICE></BEER>
    </BAR>
    <BAR type = "sushi">
            <NAME>Homma's</NAME>
            <BEER><NAME>Sapporo</NAME>
                    <PRICE>4.00</PRICE></BEER>
    </BAR> ...
</BARS>
```

XQUERY:
BBS Example – II

- Example:
  find the names of "sports bars" serving "Bud"
- FLWR Query:

```
FOR $ba IN document("bars.xml")//BAR[@type = "sports"],
WHERE $ba/BEER/[NAME = "Bud"]
RETURN <out>$ba/NAME/text()</out>;
```

NOTE 1:
$ba is assosiated with data present in the "bars.xml" file

NOTE 2:
Start at BAR nodes, i.e., select only those elements for $ba

NOTE 3:
Further reduce the number of elements to only those bars which is a "sports" bar

NOTE 4:
select only those bars from the collection $ba that have beer named "bud"

NOTE 5:
return the name of the bar

NOTE 6:
the text() function retrieves the text (name) between the name-tags inside the bar-tag

XQUERY:
BBS Example – III

- Query: find the names of "sports bars" serving "Bud"

```
1 FOR $ba IN document("bars.xml")//BAR[@type = "sports"],
2 WHERE $ba/BEER/[NAME = "Bud"]
3 RETURN <out>$ba/NAME/text()</out>;
```

- XML-file containing data (bars.xml):

```
<?XML VERSION = "1.0" STANDALONE = "no"?>
<!DOCTYPE Bars SYSTEM "bar.dtd">
<BARS>
      <BAR type = "sports"> 1
            <NAME>Joe's</NAME> 3
          2 <BEER><NAME>Bud</NAME><PRICE>2.50</PRICE></BEER>
            <BEER><NAME>Miller</NAME><PRICE>3.00</PRICE></BEER>
      </BAR>
      <BAR type = "sports"> 1
            <NAME>Mary's</NAME>
            <BEER><NAME>Miller</NAME><PRICE>3.50</PRICE></BEER>
      </BAR>
      <BAR type = "sushi">
            <NAME>Homma's</NAME>
            <BEER><NAME>Sapporo</NAME><PRICE>4.00</PRICE></BEER>
      </BAR> ...
</BARS>
```

Output:
<out>Joe's</out>

Summary

- SQL-99
  - user-defined types (UDTs) –
    both as relation and component types
  - methods for UDTs –
    user defined, observer, generator, and mutator
  - declarations
  - references –
    direct references, referencable tables, reference following
  - comparison operations – must be defined for UDTs
- XML Query example: XQUERY