ADT 2010

Introduction to

(XML, XPath &) XQuery

Chapter 10 in

Silberschatz, Korth, Sudarshan

“Database System Concepts”

Stefan Manegold
Stefan.Manegold@cwi.nl
http://www.cwi.nl/~manegold/

XML Databases

why

- *Motivation & The Big Picture: XML, DTD, XML Schema, XPath*

WHAT ← ←

- ***Crash Course XQuery***

who

- *XML files ➔ Saxon, Galax, GNU Qexo*
- *XML DBMS ➔ eXist, BerkeleyDB, MonetDB, X-Hive, Tamino, Xyleme*
- *XML RDBMS ➔ Oracle10g, SQLserver 2005, DB2*

how

- *Under The Hood of MonetDB/XQuery*
- *Some Benchmarks*

XQuery

- XQuery is currently being defined by the W3 Consortium as a standard means to query XML databases:

  `http://www.w3.org/XML/Query`

- While **XPath** forms the backbone of XQuery, some powerful **new operators** have been added, as well as an elaborated **type system**.

- XQuery is targeted to be a query language to access **database systems** with **large amounts** of data.

- Only few prototype have been presented yet, with limited functionality, e. g.
  - **Galax:** `http://db.bell-labs.com/galax/`
  - **X-Hive:** `http://www.x-hive.com/xquery`

- A full-fledged XQuery implementation is available:
  - **MonetDB/XQuery**: `http://monetdb-xquery.org/`

XQuery

- XQuery is a **functional language**.
  - XQuery expressions are **side-effect free**.
  - Expressions may be **nested** with full generality.

- Expressions **always** evaluate to **sequences**.
  - A sequence is an **ordered** collection of zero or more **items**.
  - Sequences are **flat**, never nested.

```
((1, 2), 3) = (1, (2, 3)) = (1, 2, 3)
```

  - A single item is identical to a sequence containing just that item. We call such a sequence a **singleton sequence**.

```
(42) = 42
```

  - An item can either be an **atomic value** (integer, string, . . .), or a **node**. Sequences may be **heterogeneous**, e. g.

```
(42, "foo", 4.2, <a></a>)
```

XQuery: FLWOR Expressions

- The `for-let-where-order by-return` construct (FLWOR, pronounced "flower") provides a means to operate on XQuery sequences.

```
for $p in /strip/panels/panel
  where contains  ($p/scene/text(), "Dilbert")
  return $p//bubble
```

  - The `for` construct **successively binds** each item of an expression (result of `/strip/panels/panel`) to a variable (`$p`), generating a so-called **tuple stream**.
  - This tuple stream is then **filtered** by the `where` clause, retaining some tuples, discarding others.
  - The `return` clause is evaluated **once for every tuple** still in the stream.
  - The result of the expression is an **ordered** sequence containing the **concatenated** results of these evaluations.

XQuery Example

What is the result of the following XQuery expression?

```
for $x in (1, 2, 3, 4) where $x < 4 return
  for $y in (10, 20) return
    ($x, $y)
```

XQuery Example

What is the result of the following XQuery expression?

```
for $x in (1, 2, 3, 4) where $x < 4 return
  for $y in (10, 20) return
    ($x, $y)
```

(1, 10) (1, 20) (2, 10) (2, 20) (3, 10) (3, 20)

XQuery: FLWOR Expressions

- While the `for` clause binds to each item in a sequence successively, the `let` clause binds variables to **whole sequences**:

```
let $a := //bubbles/bubble
  return count($a)
```

  - ▷ `$a` is bound to the sequence containing all `event` elements at once.

- Several `for` and/or `let` clauses may be given in a single FLWOR expression, e. g.

```
let $panels := /strip/panels,
  for $panel in $panels/panel,
    let $bubble := $panel//bubble
      where contains ($bubble/@speaker, 'dilbert')
      return ($panel/@no, count ($bubble))
```

- An **order** may be specified between `where` and `return` part using a `order by` clause.

```
for $c in //characters/character
  order by $c/text()
  return $c
```

XQuery: Element Construction

- XML **tree fragments** may be constructed "on the fly" in queries:

```
for $c in //characters/character
  return
    <person> { $c/text() } </person>
```

  - ▷ If you want to "escape" to XQuery in XML fragments, use the { / } characters.

- Element construction creates a **deep copy** of its arguments:

```
...
let $y := (<a> { $x } </a>)/child::*
  ...
```

  → $x and $y are **not** the same node!

Joins

- Joins are specified in a manner very similar to SQL

```
for $a  in  /bank/account,
    $c  in  /bank/customer,
    $d  in  /bank/depositor
where   $a/account_number = $d/account_number
    and $c/customer_name = $d/customer_name
return <cust_acct> { $c $a } </cust_acct>
```

- The same query can be expressed with the selections specified as XPath selections:

```
for $a in /bank/account
      $c in /bank/customer
$d in /bank/depositor[
                account_number = $a/account_number and
                customer_name  = $c/customer_name]
return <cust_acct> { $c $a } </cust_acct>
```

Nested Queries

- The following query converts data from the flat structure for bank information into the nested structure used in bank-1

```
<bank-1> {
    for $c in /bank/customer
    return
      <customer>
        { $c/* }
        { for $d in /bank/depositor[customer_name =
      $c/customer_name],
            $a in
      /bank/account[account_number=$d/account_number]
        return $a }
      </customer>
} </bank-1>
```

- $c/* denotes all the children of the node to which $c is bound, without the enclosing top-level tag
- $c/text() gives text content of an element without any subelements / tags

Sorting in XQuery

- The **order by** clause can be used at the end of any expression.
  E.g. to return customers sorted by name

```
for $c in /bank/customer
      order by $c/customer_name
       return <customer> { $c/* } </customer>
```

- Use **order by** $c/customer_name to sort in descending order
- Can sort at multiple levels of nesting (sort by customer_name, and by account_number within each customer)

```
<bank-1> {
    for $c in /bank/customer
    order by $c/customer_name
    return
          <customer>
          { $c/* }
          { for $d in /bank/depositor[customer_name=$c/customer_name],
                      $a in /bank/account[account_number=$d/account_number]
            order by $a/account_number
            return <account> { $a/* } </account> }
          </customer>
} </bank-1>
```

Functions and Other XQuery Features

- User defined functions

```
declare function balances($c as xs:string) as xs:decimal* {
  for $d in /bank/depositor[customer_name = $c],
      $a in /bank/account[account_number = $d/account_number]
  return $a/balance
}
```

- Types are optional for function parameters and return values
- The * (as in decimal*) indicates a sequence of values of that type

- Universal and existential quantification in where clause predicates
  - **some** $e **in** *path* **satisfies** *P*
  - **every** $e **in** *path* **satisfies** *P*
- XQuery also supports If-then-else clauses

Further Reading Material

- Easily digestable introductions to XML, XPath, and XQuery:

The Annotated XML Specification

`http://www.xml.com/axml/testaxml.htm`

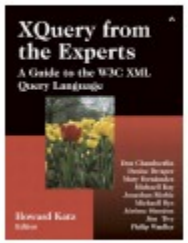
Chapter 'XPath' of 'XML in a Nutshell' (O'Reilly)

`http://www.oreilly.com/catalog/xmlnut2/chapter/`

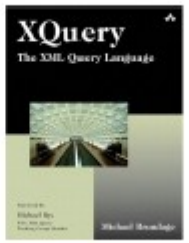
XQuery: A Guided Tour

`http://www.datadirect.com/developer/xml/xquery/docs/katz_c01.pdf`

- . . . on XPath and XQuery:



XQuery from the Experts
Jonathan Robie *et.al.*
ISBN 0-321-18060-7
Addison-Wesley, 2003



The XML Query Language
Andrew Brundage
ISBN 0-321-16581-0
Addison-Wesley, 2004