XQuery Tutorial

Agenda

- Motivation&History
- Basics
  - Literals
  - ~~XPath~~
  - FLWOR
  - Constructors
- Advanced
  - Type system
  - Functions
  - Modules

Motivation

- XML Query language to query data stores
- XSLT for translation, not for querying
- Integration language
  - Connect data from different sources
  - Access relational data represented as XML
  - Use in messaging systems
- Need expressive language for integration

Design Goals

- Declarative & functional language
  - No side effects
  - No required order of execution etc.
  - Easier to optimize
- Strongly typed language, can handle weak typing
- Optional static typing
- Easier language
  - Non-XML syntax
  - Not a cumbersome SQL extension

History

- Started 1998 as "Quilt"
- Influenced by database- and document-oriented community
- First W3C working draft in Feb 2001
  - Still not a recommendation in March 2006
  - Lots of implementations already available
  - Criticism: much too late
- Current work
  - Full text search
  - Updates within XQuery

A composed language

- XQuery tries to fit into the XML world
- Based on other specifications
    - XPath language
    - XML Schema data model
    - various RFCs (2396, 3986 (URI), 3987 (IRI))
    - Unicode
    - XML Names, Base, ID
- Think: XPath 1.0 + XML literals + loop constructs

Basics

- Literals
- Arithmetic
- ~~XPath~~
- FLWOR
- Constructors

Basics - Literals

- Strings
  - "Hello ' World"
  - "Hello "" World"
  - 'Hello " World'
  - "Hello $foo World" – doesn't work!
- Numbers
  - xs:integer - 42
  - xs:decimal – 3.5
  - xs:double - .35E1
  - xs:integer* - 1 to 5 – (1, 2, 3, 4, 5)

Literals (ctd.)

- Sequences: (1, 2.3, <foo/>, “x”)
- Construct types from strings
  - xs:QName(“f:bar”)
  - xs:float(3.4E6)
  - Identical to casting: “f:bar” cast as xs:QName
- Time values (ISO8601)
  - xs:yearMonthDuration(“PT1Y5M”)
  - xs:dayTimeDuration(“PT6D4H5M2.34S”)
  - xs:dateTime(“2006-03-24T09:30:00+01:00”)
-

Basics - Arithmetic

- Standard operators +, -, *, div, idiv
  - * can also have an XPath meaning: foo//*
  - division is “div”, not “/”
- Basic arithmetic built in functions
  - fn:sum((1, 2, 3, 4)) = 10
  - fn:ceiling(4.2) = 5, fn:floor(4.2) = 4
  - fn:round(4.5) =5
  - fn:round-half-to-even(4.45, 1) = 4.4

Basics - Comparison

- Two types of comparators
- Existential (General) comparisons
  - "=", "!=", ">", ">=", ...
  - \$X = \$Y <=> ∃ \$x ∈ \$X, \$y ∈ \$Y, \$x = \$y
  - Relaxed typing (e.g. <x>5</x> = 5)
- Value comparisons
  - "eq", "neq", "gt", "ge", ...
  - Enforces exactly one element on each side and matching types (error otherwise)

Basics – Boolean Stuff

- Built in type xs:boolean
  - Construct using xs:boolean(“true”)
  - valid literals: “true”, “false”, “0”, “1”
  - easier: fn:true() and fn:false()
- Boolean operators
  - true() and true(), false() or true(), not(true())
- Effective boolean value
  - if (<x/>) is true
  - if (“asd”) is true, if (“”) is false
  - if (5) is true, if (0) is false

Basics - Conditionals

- if-then-else

```
if (5 = 2) then "WTF?"
else "Yeah"
```

Text

- Else is always needed (functional!)
  - Use empty sequence ()

```
if ($mycond) then "foo"
else ()
```

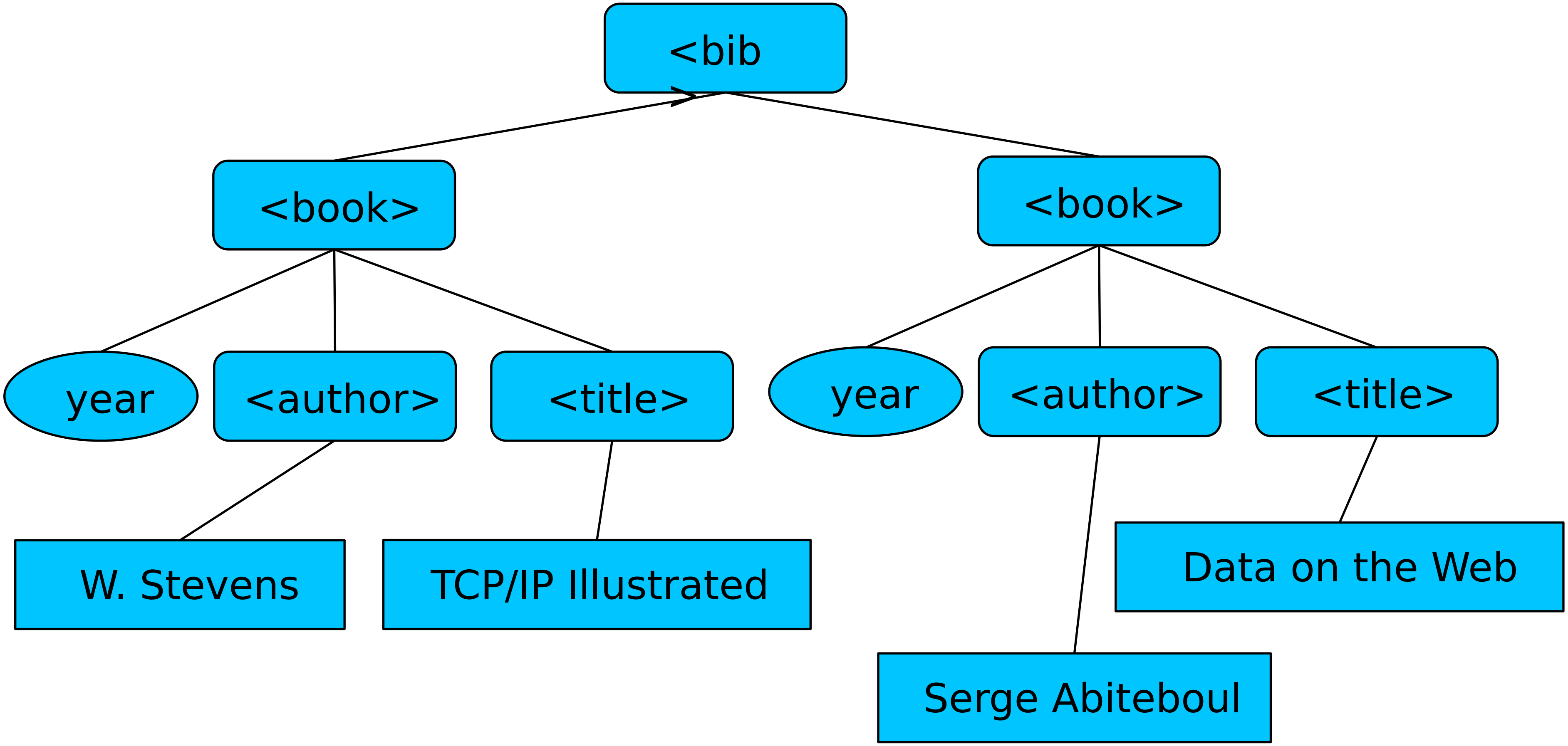
Basics – The Prolog

- Comes in front of the query
- Declare namespaces, global variables, global options, external variables etc.
- Important declarations
  - declare namespace foo = “http://bar”;
- Predefined namespace prefixes
  - xml, xmlns, fn, xs, xsi, op, xdt, local
  - plus implementation defined prefixes (e.g. xhive)

Basics - XPath

- Express path “patterns” on XML trees
  - doc('foo.xml')/a/b/c
- Each step
  - results in sequence of nodes
  - (Conceptually) sorts nodes in document order
  - de-duplicates nodes

XPath Example Doc



XPath Axes

- Abbreviated
  - /bar = /child::bar
  - //bar = /descendant-or-self::node()/child:bar
  - /@x = /attribute::x
  - /../bar = /parent::node()/child::bar
- All directions
  - parent::, self::, child::, attribute::
  - descendant::, descendant-or-self::
  - ancestor::, ancestor-or-self::
  - preceding-sibling::, following-sibling::, preceding::,

Node tests

- After each axis, write a node test
- Pseudo-functions
  - item(), node(), element(), attribute(), text() document-node(), processing-instruction(), comment ()
- Qualified names, wildcards
  - /foo:bar, /*, /*:bar, /foo:*
- Weird stuff you won't need
  - element(foo:*, xs:string)

XPath Predicates

- Filter node sequences from steps
  - /foo//bar[@attr = 42]
- Filter by position: /foo[3]
- Special functions
  - foo[position() > 3]
  - foo[last()]
- Fully composable:

Text

```
/foo[.//bar[@attr = 42]
     and count(for $x in
                     doc('x.xml')//x
               where $x/@y = 'abc'
               return $x) > 3]
```

The doc() function

- Used to access documents
- Parameter is a string containing a URI
  - doc('foo.xml')
  - doc('/bar/foo.xml')
  - doc('http://www.example.com/foo.xml')
  - doc('xhive://foo/bar/../test.xml')
- Accessing a library (doc('lib/')) gives
  - all documents in the library
  - all documents in descendant libraries

Basics - FLWOR Expressions

- Pronounced “flower”

F	– for
L	– let
W	– where
O	– order by
R	– return

for expressions

- Iterate over all elements in a sequence
- Bind current element to a variable
- Trivial example:

```
for $x in /foo/bar
return $x
```

- 100% identical to simply /foo/bar

for for joining

- for statements are great for joins

```
for $x in doc('foo.xml')//x,
    $y in doc('bar.xml')//y
where $x/@ref = $y/@name and $x/age > 42
return $y
```

- Optimizable: this may or may not be executed as a nested loop

More for

- Variables can be in namespaces

```
declare namespace pre = "http://foo/bar";
for $pre:x in /foo/bar
return $pre:x
```

- Can bind an index variable

```
for $x at $i in /foo/bar
return $x
```

let and where expressions

- let: Bind a whole sequence to a variable
- where: filter results

```
let $docs := doc('foo.xml')[root/@usecount > 5]
for $doc in $docs/root/document
where $doc/@name = 'mydoc'
return doc($doc/href)
```

- Careful:

```
 let $x := //foo
 where $x/@attr = 5
 return $x
```

this means: return all //foo if any of them meets $x/@attr = 5

WARNING: Immutable variables

- XQuery is functional
  - variables are immutable
  - if a variable goes out of scope, it's reset
  - Query below will create a series of "2" and <book/> elements

```
(: This doesn't work! :)
let $i := 1
for $x in //book
let $i := $i + 1
return ($i, $x)
```

Order by

- Order the results of the whole FLWOR expression

```
(: Get all the books newer than 1990 whose author has
 : written more than two other books, order by
 : the name of the first author
 :)
let $bib := doc('bib.xml')
for $book in $bib/bib/book
let $authorcount := count(
  $bib/bib/book[author = $book/author]) - 1
where $book/@year > 1990
  and $authorcount > 2
order by $book/author[1]
return $book
```

Constructors

- Easy way to construct XML within XQuery
- Nearly 1-1 compatible with real XML
- Two syntaxes
  - direct constructors are literal XML
  - computed constructors are descriptions

Direct constructor example

- Contents of constructors are copied into the tree

```
<bib>
{
  for $book in doc('bib.xml')//book,
      $review in doc('reviews.xml')//review
  return
    <bookreview year="{ $book/@year }">
      { $book/title, $book/price, $review/text }
    </bookreview>
}
</bib>
```

Escaping

- Escaping in constructors by duplication
  - `{ : <foo>{{</foo>`
  - `' : <foo attr='x''y'>{{</foo>`
  - `“ : <foo attr=”x””y”>{{</foo>`
- Or as entities
  - `&apos;, &quot;`

Computed constructors

- Useful for
    - Elements with dynamic name
    - document constructors
    - processing instructions, comments

```
for $elem in //elem
return
  document {
    for $pi in $elem//pi
    return
      processing-instruction { $pi } { $pi/content },
      comment { $elem/comment },
      element foo { $elem/node() }
  }
```

Advanced

- So much for the basic part
- Type System
- Functions
- Modules

Type System

- Basic type in XQuery: The Sequence
- Sequences
    - can be of any length (including 1)
    - can contain atomic values, e.g. numbers or QNames
    - can contain non-atomic values, e.g. XML
- Atomic vs. Non-atomic
    - fn:data() “atomizes” XML

Type System (ctd.)

- Types can be specified on
  - FLWOR parts
  - (external) functions
  - (external) variables
- Syntax similar to node tests in paths
- Cardinality of a sequence
  - * - any
  - + - at least one
  - ? - one, optional
  - no sign means always exactly one

Type System (ctd.)

- Examples:
  - element(foo)* - Sequence of XML elements with QName “foo”
  - xs:integer - A single integer value
  - xs:QName? - optional xs:QName
  - attribute(*, xs:IDREF)*
    - any amount of attributes with any QName that have the type xs:IDREF
  - xs:string+ - At least one xs:string

Types in FLWORs

- Specify types with “as” keyword
- Types are only checked if the variable is actually used (lazily)

```
let $x as element(foo) := //foo
for $y as xs:integer in data($x//nums)
return $y
```

- Alternatively: static typing
  - pessimistic static typing
  - few implementations available
  - completely unusable without XML Schemas

Casting and typeswitch

- Use “cast as” to convert between types

```
"5" cast as xs:double
```

- Use typeswitch for dynamic typechecking

```
typeswitch ($x)
  case $y as xs:integer return "integer"
  case $y as xs:double return "double"
  case $y as element() return "XML element"
  default $y return "Unknown type"
```

Validation

- Import XML Schema into query scope
- Validate results of expressions against XML schema
- Check against and cast to user defined types

```
import schema namespace foo = 'http://bar' at 'foo.xsd';
validate strict { <foo:bar>Hello World!</foo:bar> }
```

Functions

- User defined functions in addition to the function library (beware of semicolon!)

```
declare function local:myfunc($x as element())
  as xs:integer
{
  let $nums := root($x)//*[@ref = $x/ref]/mass
  return sum($nums)
}
```

- Recursion is allowed

```
declare function local:sum($start as xs:integer,
                           $acc as xs:integer)
  as xs:integer
{
  if ($start eq 0) then $acc
  else local:sum($start - 1, $acc + $start)
}
```

Modules

- Group XQuery statements into modules
- Modules export
  - Global variables (declare variable $x:y := ...)
  - Functions
- Modules have a prolog only, no body
- Declare modules using
  module namespace x = “http://...”;
- Import modules into queries using
  import module namespace x = “http://...”
  at “/modules/x.xq”;

The specification

- XQuery spec is divided into
  - Requirements, Use Cases
  - Main specification
  - Functions and Operators
  - Data model
  - Serialization
  - Formal semantics
- Goal: modularization, clear scope/requirements, unambiguous semantics (contrast XML Schema)
- Large parts shared with XSLT 2.0 / XPath 2.0

State of the spec

- XQuery spec is a „Proposed Recommendation“
- will be promoted to full Recommendation „real soon now“
- ~50 implementations known
- Test suite with > 10.000 tests
  - results from 14 implementations known
  - 10 implementations over 98% correct
- Extensions for Full Text Search and Updates in progress

Further reading

- Specs and other documents
  http://www.w3.org/XML/Query/
- Introduction by Michael Kay
  http://www.stylusstudio.com/xquery_primer.html
- X-Hive/DB
  http://www.x-hive.com/products/db/