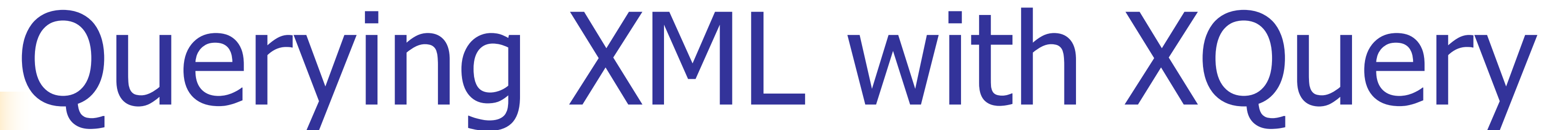
© Copyright 2003, DataDirect Technologies



# Querying XML with XQuery

Jonathan Robie

DataDirect Technologies

jonathan.robie@datadirect.com

Colorado Software Summit

© Copyright 2003, DataDirect Technologies

# Basic Queries

© Copyright 2003, DataDirect Technologies

# XML Query History

- Early query facilities for SGML
- 1998: "Roll your own query language"
- Feb 1998: XQL proposal
  - http://metalab.unc.edu/xql
- Aug 1998: XML-QL submission
  - http://www.w3.org/TR/NOTE-xml-ql/
- Dec 1998: W3C QL'98 Workshop
  - http://www.w3.org/TandS/QL/QL98
- Nov 1999: XPath Recommendation
  - http://www.w3.org/TR/xpath

© Copyright 2003, DataDirect Technologies

# XQuery Working Drafts

- XML Query Requirements
  http://www.w3.org/TR/xquery-requirements/
- XML Query Use Cases
  http://www.w3.org/TR/xquery-use-cases/
- XQuery 1.0: An XML Query Language
  http://www.w3.org/TR/xquery/
- XML Query 1.0 and XPath 2.0 Data Model
  http://www.w3.org/TR/xpath-datamodel/
- XSLT 2.0 and XQuery 1.0 Serialization
  http://www.w3.org/TR/xslt-xquery-serialization/
- XQuery 1.0 and XPath 2.0 Functions and Operators
  http://www.w3.org/TR/xpath-functions/
- XML Query Formal Semantics
  http://www.w3.org/TR/xquery-semantics/

Colorado Software Summit

© Copyright 2003, DataDirect Technologies

# XML Query Data Model

- Joint with XPath 2.0, XSL 2.0
- Ordered, labeled forest
- Based on XML Information Set, PSVI
  - – but much leaner!
- Seven kinds of nodes
  - document, element, attribute, text, namespace, processing instruction, and comment
  - Nodes have identity

© Copyright 2003, DataDirect Technologies

# Document Node *vs.* Root Element

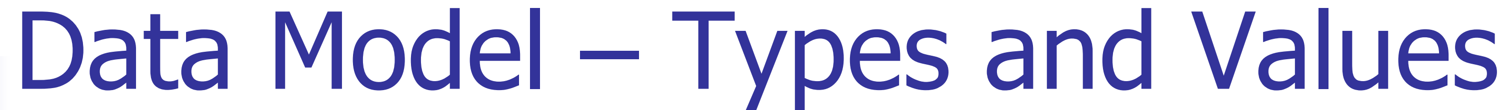
```
<?xml version="1.0"?>
<?format eyesight="20/200"?>

<!-- You can't see the document node in this file. -->

<!– These children of the document node may be:
     1. comments
     2. processing instructions
     3. the root element  -->

<rootElement>Hi, Mom!</rootElement>
```

# Document Order

- "First character order"
  - Document node first
  - Element nodes before their children
  - Siblings in order of occurrence
  - Attributes before child elements – but in any order
  - Regions of uninterrupted text are treated as 'text nodes'

<[2]book y[3]ear="1994">
  <[4]title>T[5]CP/IP Illustrated</title>
  <[6]author>
    <[7]last>S[8]tevens</last>
    <[9]first>W[10].</first>
  </author>
</book>

© Copyright 2003, DataDirect Technologies

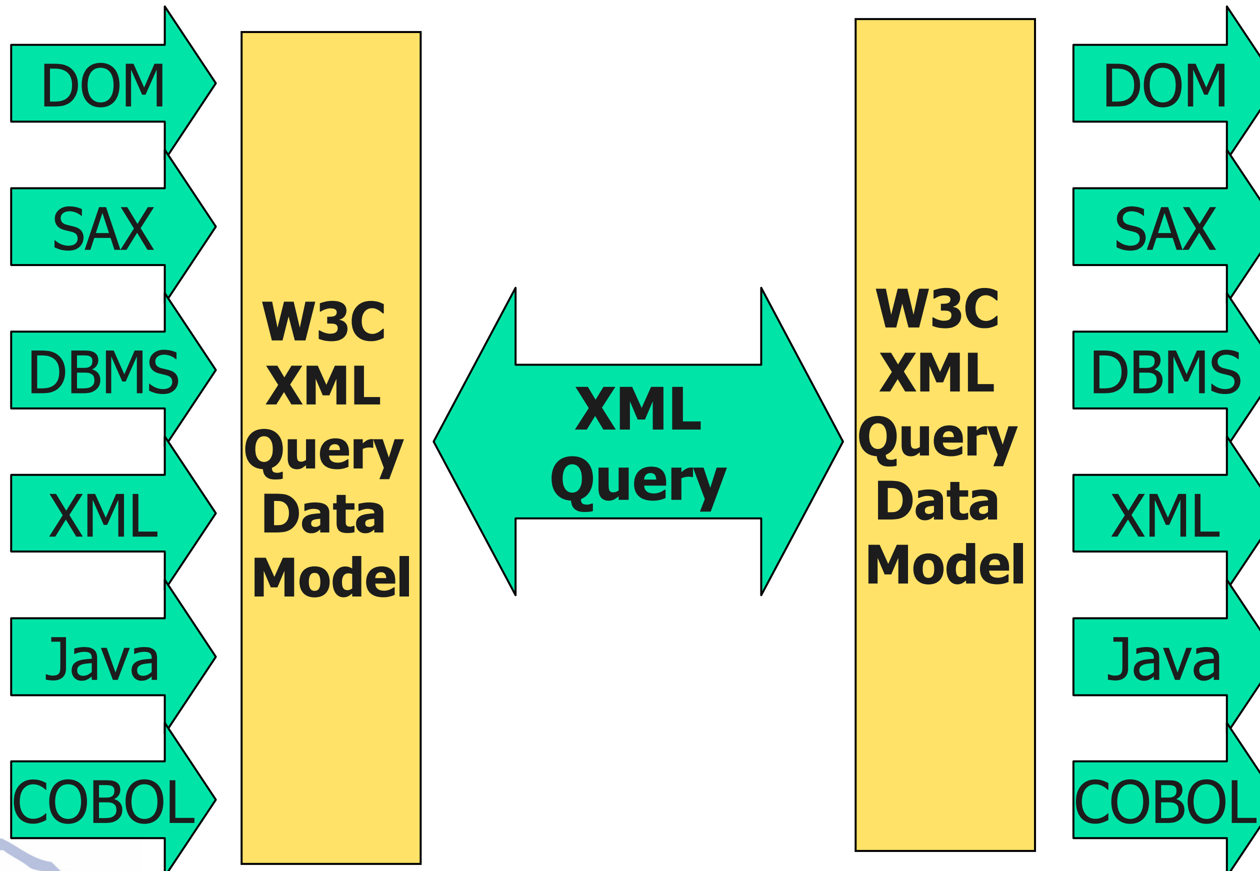


# Data Model – Types and Values

- Classes of documents
  - Well-formed documents
  - DTD-valid documents
  - W3C XML Schema-validated documents
- Two interpretations of a node
  - typed value – with XML Schema type
  - string value – lexical representation
- Untyped data is xs:anyType or xs:anySimpleType
  - Failed validation
  - No validation attempted

© Copyright 2003, DataDirect Technologies

# XQuery and the Data Model



© Copyright 2003, DataDirect Technologies

# XML Query Formal Semantics

- Static Semantics
  - Type inference rules
  - Structural subsumption
- Dynamic Semantics
  - Value inference rules
  - Define the meaning of XQuery expressions in terms of the XML Query Data Model

© Copyright 2003, DataDirect Technologies

# The XQuery Language

- XQuery is a functional language
  - Evaluating expressions – not performing commands.
  - A query is an expression
  - Expressions can be nested with full generality.
  - A functional language - but syntax often does not look functional.
- Based on OQL, SQL, XQL, XML-QL, XPath

© Copyright 2003, DataDirect Technologies

# XQuery Expressions

- Element constructors
- Path expressions
- Restructuring
  - Sorting
  - FLWR expressions
  - Conditional expressions
  - Quantified expressions
- Operators and functions
- List constructors
- Expressions that test or modify datatypes

© Copyright 2003, DataDirect Technologies

# Basics

© Copyright 2003, DataDirect Technologies

# Literals and Data Types

- Comments
  - (: doesn't this look cheerful? :)
- W3C XML Schema simple types
- String
  - "a string"
  - 'a string'
  - "This is a string, isn't it?"
  - 'This is a "string"'
  - "a "" or a ' delimits a string literal"
  - 'a " or a " delimits a string literal'

© Copyright 2003, DataDirect Technologies

# Character Entities and Strings

- Predeclared character entities

| < | < |
|---|---|
| > | > |
| & | & |
| " | " |
| ' | ' |

- Using character entities in strings

```
'&lt;bold&gt;A sample element.&lt;/bold&gt;'
```

© Copyright 2003, DataDirect Technologies

# Literals and Data Types

- Numeric Literals
  - 1 (: An integer :)
  - -2 (: An integer :)
  - +2 (: An integer :)
  - 1.23 (: A decimal :)
  - -1.23 (: A decimal :)
  - 1.2e5 (: A double :)
  - -1.2E5 (: A double :)

© Copyright 2003, DataDirect Technologies

# Input functions

- doc() returns an entire document, identifying the document by a URI. To be more precise, it returns the document node.
- collection() returns a collection, which is any sequence of nodes that is associated with a URI. This is often used to identify a database to be used in a query.

© Copyright 2003, DataDirect Technologies

# Built-in Functions

- The following are used in this tutorial:
  - max(), min(), sum(), count(), avg()
  - distinct-values(), empty(), exists()
  - collection(), doc()
  - contains(), concat()
  - QName()
  - position(), last()
- Described more fully as encountered
- Many more in Functions & Operators spec.

Colorado Software Summit

© Copyright 2003, DataDirect Technologies

# Element Constructors

© Copyright 2003, DataDirect Technologies

# XML Element Constructors

```
(: These constructors look like the XML they construct :)
    <book year="1994">
      <title>TCP/IP Illustrated</title>
      <author>
        <last>Stevens</last>
        <first>W.</first>
      </author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
    </book>
```

# Expressions in XML Constructors

- XQuery expressions are made to be combined
- Angle brackets escape to XML syntax
- Curly braces escape to XQuery expression syntax

```
(: the year contains two subtractions! :)
<book year="{2002-4-4}">
    <title>TCP/IP Illustrated</title>
    <publisher>Addison-Wesley</publisher>
    <price>{ 70-5-0.05 }</price>
</book>
```

© Copyright 2003, DataDirect Technologies

# Computed Constructor Syntax

```
element book {
   (: Attributes must appear first! :)
   attribute year { 1994 },
   element title { "TCP/IP Illustrated" },
   element author {
       element last { "Stevens" },
       element first { "W." }
   },
   element publisher { "Addison-Wesley" },
   element price { 65.95 }
}
```

© Copyright 2003, DataDirect Technologies

# Computing Names Dynamically

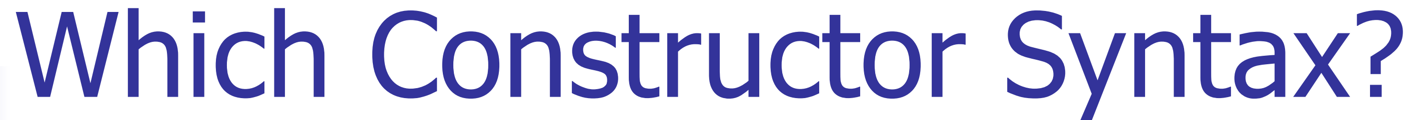
- Query

```
let $b := concat("book", "001")
return
    element { $b } { $b }
```

- Output:

```
<book001>book001</book001>
```

© Copyright 2003, DataDirect Technologies



# Which Constructor Syntax?

- XML constructor syntax looks like XML
  - Nothing new to learn
  - Easy to distinguish from rest of query
- Either syntax works for most construction
  - Matter of taste
  - Most people use XML constructor syntax
- Computed constructor syntax allows names to be computed at run time

© Copyright 2003, DataDirect Technologies

# Other Constructors

- document { … }
  - Creates a document node
  - See next slide…
- text { … }
  - Creates a text node
  - Used for untyped character data
  - Same as text in an element (when datatypes are not being used)

© Copyright 2003, DataDirect Technologies

# Creating a Document Node

```
document {
   <?format eyesight="20/200"?>

   <!-- You can't see the document node in this file. -->

   <!– These children of the document node may be:
         1. comments
         2. processing instructions
         3. the root element  -->

   <rootElement>Hi, Mom!</rootElement>
}
```

© Copyright 2003, DataDirect Technologies

# Practice: Element Constructors

- Ensure that XQuery is properly installed.
- Create the following element using XML syntax:

```
<product id="123">
  <name>Left-handed smoke shifter</name>
  <price>16.34</price>
  <description>Finally, a shifter for left-
   handers!</description>
</product>
```

© Copyright 2003, DataDirect Technologies

# Practice: Element Constructors

- Create the same element, computing the numeric values with arithmetic expressions.
- Create the same element using computed element constructor syntax.
- Create the same element using computed element constructor syntax, creating the element name "product" at run time.

© Copyright 2003, DataDirect Technologies

# Path Expressions

© Copyright 2003, DataDirect Technologies

# Path Expressions

- Start with an input function or a variable
  - In some implementations, you can start with a leading / - what it is bound to is implementation-defined
  - In some implementations, variables may be predeclared
- Navigate using step expressions
- Filter using predicates (part of a step expression)

© Copyright 2003, DataDirect Technologies

- One parameter: a URI
  - doc("books.xml")
  - doc("http://www.eg.com/books.xml")
- URI identifies a document
  - Returns document node
  - Not same as document element!

© Copyright 2003, DataDirect Technologies

- One parameter: a URI
  collection("jdbc:ddt:sql://host:1433;database='northwind'")

- Returns a sequence of nodes from a collection
- Collections are vendor-defined, identified by URI

# Navigation: Sample Input Document

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price> 65.95</price>
  </book>
```

© Copyright 2003, DataDirect Technologies

# Navigating with Path Expressions

- Single slash, name test
  - doc("bib.xml")/bib
  - "all bib elements that are children of document node"
- Double slash
  - doc("bib.xml")//book
  - "all books elements that are descended from the document node"
- Predicates
  - doc("bib.xml")//book[@year="1994"]
  - "all books elements, containing a year attribute equal to '1994', that are descended from the document node"

Colorado Software Summit

© Copyright 2003, DataDirect Technologies

# Numeric Predicates

- First book:
  - input()/bib/book[1]
- First author of each book:
  - input()/bib/book/author[1]
  - input()//author[1]
- First author in document:
  - (input()/bib/book/author)[1]
  - (input()//author)[1]

© Copyright 2003, DataDirect Technologies

# Step Expressions

- Single slashes separate "steps":
  - doc("bib.xml") / child::bib[ count(child::book) > 3]
- Basic components of a step:
  - Context node – sets the context for the step
  - Axis – states the direction of the step relative to context node (child, descendant, attribute, self, descendant-or-self, parent)
  - NodeTest – returns nodes that satisfy a condition on their name (NameTest) or kind (KindTest)
  - Predicates – filters nodes based on a condition

© Copyright 2003, DataDirect Technologies

- The "axis" determines the direction in which a step navigates from the context node
- XQuery normally uses "abbreviated syntax" for axes. Single slash means child axis:

```
doc("bib.xml")/bib
=>
doc("bib.xml")/child::bib
```

- Possible axes in XQuery: child, descendant, attribute, self, descendant-or-self, parent
- Explicit use of axes rare in XQuery

© Copyright 2003, DataDirect Technologies

# Steps: Context Nodes and Evaluation

- Context nodes are drawn from the expression on the left of the slash
- LeftExpr / RightExpr
  - Evaluate LeftExpr – must be a sequence of nodes, or error is raised.
  - Using each node from LeftExpr as a "context node"; evaluate RightExpr – must be a sequence of nodes, or error is raised.
  - Merge results for each context node; sort in document order (which eliminates duplicates).
- Examples:
  - doc("bib.xml")/child::bib/child::book/child::author
  - doc("bib.xml")/descendant::book/child::author[1]

Colorado Software Summit

© Copyright 2003, DataDirect Technologies

# Referring to the Context Node

- In path expressions, "." refers to the context node.
- The self axis also refers to the context node.
- The following queries are equivalent:
  - doc("bib.xml")/bib
  - doc("bib.xml")/bib/.
  - doc("bib.xml")/bib/self::*

© Copyright 2003, DataDirect Technologies

# Steps: Two Kinds of NodeTest

- NameTest
  - Element name: 'bib' matches bib elements
  - Attribute name: '@bib' matches bib attributes
  - Wildcard:
    - '*' matches any element
    - '@*' matches any attribute
- KindTest
  - node() matches any node
  - comment() matches any comment
  - text() matches any text node
  - processing-instruction() matches any pi
  - processing-instruction("target") matches pi for 'target'

© Copyright 2003, DataDirect Technologies

# Namespaces and NameTest

- Declaring and using prefixes

```
declare namespace xhtml ="http://www.w3.org/1999/xhtml"
doc("aspect.xhtml")//xhtml:table
```

- Matching based on URI, not prefix

© Copyright 2003, DataDirect Technologies

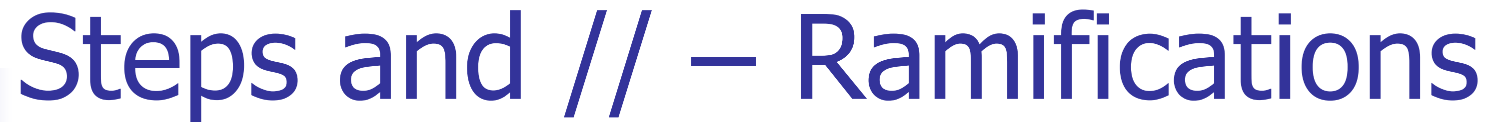
# Steps and //

- Abbreviated notation is defined using axis notation
- Formal definition of //:
  - /descendant-or-self::node()/
  - a//b has three steps!
- Examples:
  - doc("bib.xml")//book

  => doc("bib.xml")/descendant-or-self::node()/child::book
  - doc("bib.xml")//@year

  => doc("bib.xml")/descendant-or-self::node()/attribute::year

© Copyright 2003, DataDirect Technologies



# Steps and // – Ramifications

- Context node comes from the middle step – not expression on left of //
- position() is relative to context node
- First author *in any given* element:
  - input()//author[1]

  => input()/descendant-or-self::node()/child::author[1]

© Copyright 2003, DataDirect Technologies

# Predicates – Common Patterns

- Condition on element value

```
doc("bib.xml")//author[last="Stevens"]
```

- Condition using function

```
doc("bib.xml")//book[count(author)>2]
doc("bib.xml")//book[contains(title, "Web")]
```

- Combining conditions

```
doc("bib.xml")//author[last="Stevens" and first="W."]
doc("bib.xml")//author[last="Stevens"][first="W."]
doc("bib.xml")//author[last="Stevens" or first="W."]
```

© Copyright 2003, DataDirect Technologies

# Predicates – Numeric Predicates

- position() is the position of the context item:

```
doc("bib.xml")//author[3]
doc("bib.xml")//author[position()=3]
```

- last() is the length of the sequence containing context item:

```
doc("bib.xml")//author[last()]
doc("bib.xml")//author[position()=last()]
```

- Expressions using position:

```
doc("bib.xml")//author[position() mod 2 = 1]
doc("bib.xml")//author[(position()>2) and (position()<5)]
```

© Copyright 2003, DataDirect Technologies

# Path Expressions in Constructors

- XQuery expressions are designed to be easily combined.
- Combining path expressions and element constructors:

```
<publishers count="{count(doc("bib.xml")//publisher)}">
    <head> Publishers </head>
    { doc("bib.xml")//publisher }
</publishers>
```

© Copyright 2003, DataDirect Technologies

© Copyright 2003, DataDirect Technologies

# FLWOR Expressions

- for - let - where - order by - return
- Similar to SQL's SELECT - FROM - WHERE

```
for $book in doc("bib.xml")//book
let $title := $book/title
where $book/publisher = "Addison-Wesley"
order by $title
return
  <book>
      {
     $title,
        $book/author
      }
  </book>
```

© Copyright 2003, DataDirect Technologies

# for *vs.* let

- for iterates on a sequence, binds a variable to each node
- let binds a variable to a sequence as a whole

```
for $book in doc("bib.xml")//book
let  $a := $book/author
where contains($book/publisher, "Addison-Wesley")
return
  <book>
        {
    $book/title,
        <count> Number of authors: { count($a) } </count>
        }
  </book>
```

Colorado Software Summit

© Copyright 2003, DataDirect Technologies

# Conditional Expressions

- IF expr THEN expr ELSE expr

```
for $h in input()//holding
return
      <holding>
           {
             $h/title,
             if ($h/@type = "Journal")
                then $h/editor
                else $h/author
           }
      </holding>
```

© Copyright 2003, DataDirect Technologies

# Inner Joins

```
for $book in doc("www.bib.com/bib.xml")//book,
    $quote in doc("www.bookstore.com/quotes.xml")//listing
where $book/isbn = $quote/isbn
order by $book/title
return
        <book>
                { $book/title }
                { $quote/price }
        </book>
```

© Copyright 2003, DataDirect Technologies

# Outer Joins

```
for $book in doc("bib.xml")//book
order by $book/title
return
        <book>
          { $book/title }
          {
             for $review in doc("reviews.xml")//review
             where $book/isbn = $review/isbn
             return $review/rating
           }
        </book>
```

© Copyright 2003, DataDirect Technologies

# Quantifiers

- EVERY var IN expr SATISFIES expr
- SOME var IN expr SATISFIES expr

```
for $b in input()//book
   where every $p in $b//para satisfies
        contains($p, "sailing")
        and contains($p, "windsurfing")
   return $b/title
```

# Transformations – Bibliography

```
<?xml version="1.0"?>
<bib>
  <book>
    <title> Harold and the Purple Crayon </title>
    <author>
       <lastname> Johnson </lastname>
       <firstname> Crockett </firstname>
    </author>
    <pubinfo>
       <publisher> Harper and Row </publisher>
       <price> 4.76 </price>
       <year> 1995 </year>
    </pubinfo>
  </book>
</bib>
```

© Copyright 2003, DataDirect Technologies

# Books by Author

```
<?xml
<bib>
  <bo
    <ti
    <au

    </a
    <p

    </p
  </bo
</bib>
```

```
<booksByAuthor>
   <author>
      <name>
         <last> Johnson </last>
         <first> Crockett </first>
      </name>
    <title> Harold and the Purple Crayon </title>
    <title> Harold's Fairy Tale </title>
    <title> Harold and the Circus </title>
    <title> Harold's ABC's </title>
 <author>
   . . .
</booksByAuthor>
```

# Removing Duplicates

- distinct-values()

```
distinct-values(doc("bib.xml")//publisher)
```

```
<publisher>Addison-Wesley</publisher>
<publisher>Morgan Kaufmann Publishers</publisher>
<publisher>Kluwer Academic Publishers</publisher>
```

- Extracts distinct atomic values:

```
Addison-Wesley
Morgan Kaufmann Publishers
Kluwer Academic Publishers
```

© Copyright 2003, DataDirect Technologies

# Inverting the Hierarchy

```
<results>
 {
  let $a := doc("data/xmp-data.xml")//author
  for $last in distinct-values($a/last),
      $first in distinct-values($a[last=$last]/first)
  return
     <result>
        { $last, $first }
        {
           for $b in doc("data/xmp-data.xml")/bib/book
           where some $ba in $b/author satisfies ($ba/last = $last and $ba/first=$first)
           return $b/title
           sort by (title)
        }
     </result>
  sort by (last, first)
 }
</results>
```

© Copyright 2003, DataDirect Technologies

# Combining Sequences

- Union
  - Combines two sequences, eliminates duplicates, returns in doc order
  - Two syntaxes
  - Examples

```
doc("data/xmp-data.xml")//(author | editor)
doc("data/xmp-data.xml")//(author union editor)
```

- Intersection
  - Returns intersection of two sequences in document order
  - Example

```
author/last intersect editor/last
```

© Copyright 2003, DataDirect Technologies

# Combining Sequences *(Continued)*

- Except
  - Removes items from a sequence
  - Handy for editing elements
  - Example:

```
let $b := doc("data/xmp-data.xml")//book[1]
return $b/* except $b/author
```

© Copyright 2003, DataDirect Technologies

# SQL-like Queries

© Copyright 2003, DataDirect Technologies

# A Relational View

| USERS | USERID | NAME | RATING |
| --- | --- | --- | --- |

| ITEMS | ITEMNO | DESCRIPTION | OFFERED_BY | RESERVE_PRICE |
| --- | --- | --- | --- | --- |

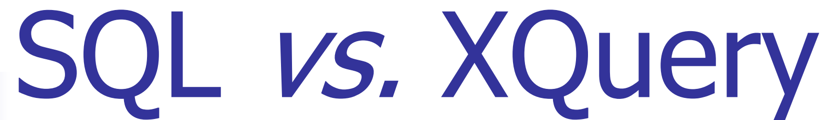
| BIDS | USERID | ITEMNO | BID_AMOUNT | BID_DATE |
| --- | --- | --- | --- | --- |

```
<users>
   <row>
      <userid>
       1243
      </userid>
      <name>
       humphrey
      </name>
      <rating>
```

```
<items>
   <row>
      <itemno>
        1066
      </itemno>
      <description>
        unicycle
      </description>
      <offered_by>
```

```
<bids>
   <row>
      <userid>
        1243
      </userid>
      <itemno>
        1066
      </itemno>
      <bid_amount>
```

© Copyright 2003, DataDirect Technologies




"Find item numbers of Bicycles"

- SQL:

```
SELECT itemno
FROM items AS i
WHERE description LIKE 'Bicycle'
ORDER BY itemno;
```

- XQuery:

```
for $i in doc("items.xml")//item_tuple
order by $i/itemno
where contains($i/description, "Bicycle")
return $i/itemno
```

© Copyright 2003, DataDirect Technologies

# Let and Aggregates

"List item numbers that have more than 10 bids, and their bid counts"

- SQL:

```
SELECT itemno, count(*) AS bid_count
FROM bids
GROUP BY itemno
HAVING count(*) > 10
ORDER BY bid_count DESC;
```

- **XQuery:**

```
for $i in distinct-values(doc("bids.xml")//bid_tuple/itemno)
let $count := count(doc("bids.xml")//bid_tuple[itemno = $i])
order by $count descending
where $count > 10
return <popular_item count="{$count}">{ $i }</popular_item
```

© Copyright 2003, DataDirect Technologies

# Inner Join

"List names of users and descriptions of the items they offer"

- SQL:

```
SELECT u.name, i.description
FROM users AS u, items AS i
where u.userid = i.offered_by
ORDER BY name, description;
```

- XQuery

```
for $u in doc("users.xml")//user_tuple,
    $i in doc("items.xml")//item_tuple
where $u/userid = $i/offered_by
order by $u/name, $i/description
return
 <offering> {
    $u/name,
    $i/description
  } </offering>
```

© Copyright 2003, DataDirect Technologies

# Outer Join

"List names of users and descriptions of the items they offer, including users who have not offered any items"

- SQL:

```
SELECT u.name, i.description
FROM users u
   LEFT OUTER JOIN items i
   ON u.userid = i.offered_by
ORDER BY u.name, i.description
```

- XQuery

```
for $u in doc("users.xml")//user_tuple
order by $u/name
return
 <seller>
   {
    $u/name,
    for $i in doc("items.xml")//item_tuple
    where $u/userid = $i/offered_by
    order by $u/description
    return $i/description
   }
 </seller>
```

© Copyright 2003, DataDirect Technologies

# Queries with Positional Variables

```
for $t at $i in doc("books.xml")//title
return <title pos="{$i}">{string($t)}</title>
```

```
<title pos="1">TCP/IP Illustrated</title>
<title pos="2">Advanced Programming in the Unix
   environment</title>
<title pos="3">Data on the Web</title>
<title pos="4">The Economics of Technology and Content
   for Digital TV</title>
```

© Copyright 2003, DataDirect Technologies

# Position May Convey Meaning

| **Title** | **publisher** | **price** | **year** |
|---|---|---|---|
| TCP/IP Illustrated | Addison-Wesley | 65.95 | 1994 |
| Advanced Programming in the Unix environment | Addison-Wesley | 65.95 | 1992 |
| Data on the Web | Morgan Kaufmann Publishers | 39.95 | 2000 |
| The Economics of Technology and Content for Digital TV | Kluwer Academic Publishers | 129.95 | 1999 |

© Copyright 2003, DataDirect Technologies

# Position May Convey Meaning
*(Continued)*

```
<table border="1">
  <thead>
    <tr>
      <td>title</td>
      <td>publisher</td>
      <td>price</td>
      <td>year</td>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>TCP/IP Illustrated</td>
      <td>Addison-Wesley</td>
      <td>65.95</td>
      <td>1994</td>
    </tr>
    <tr>
      <td>Advanced Programming in the Unix environment</td>
      <td>Addison-Wesley</td>
      <td>65.95</td>
      <td>1992</td>
```

© Copyright 2003, DataDirect Technologies

# Extracting XML from HTML Table

```
let $t := doc("bib.xhtml")//table[1]
for $r in $t/tbody/tr
return
  <book>
   {
     for $c at $i in $r/td
     return element{ data($t/thead/tr/td[$i]) }
                        { string( $c) }
   }
  </book>
```

© Copyright 2003, DataDirect Technologies

# Output from Previous Query

```
<book>
   <title>TCP/IP Illustrated</title>
   <publisher>Addison-Wesley</publisher>
   <price>65.95</price>
   <year>1994</year>
</book>
<book>
   <title>Advanced Programming in the Unix environment</title>
   <publisher>Addison-Wesley</publisher>
   <price>65.95</price>
   <year>1992</year>
</book>
```

© Copyright 2003, DataDirect Technologies

# Arithmetic, Functions, Comparisons

© Copyright 2003, DataDirect Technologies

# Arithmetic

- Binary operators: +, -, *, div, idiv, mod
- Unary operators: +, -
- "Atomization" extracts values from nodes

  4 * <foo>5</foo>

- If an operand is untyped, it is cast to double (see above example)
- If an operand is empty sequence, result is empty sequence (c.f. SQL nulls).

© Copyright 2003, DataDirect Technologies

- If the given value is already a single atomic value or an empty sequence, return the given value.
- If the given value is a single node, the typed value of the node is extracted and returned; if the result is a sequence containing more than one item, a type exception is raised.
- In any other case, atomization raises a type exception.

| Value Comparison | General Comparison |
|---|---|
| `eq` | = |
| `ne` | != |
| `lt` | < |
| `le` | <= |
| `gt` | > |
| `ge` | >= |

© Copyright 2003, DataDirect Technologies

# Value Comparisons

- Nostalgic FORTRAN syntax
- Compares two single values:

```
for $b in doc("books.xml")//book
where $b/title eq "Data on the Web"
return $b/price
```

- Atomization is applied to operands
- Error if not single values
- Does not cast untyped data:

```
(: !! Raises an error !! :)
for $b in doc("books.xml")//book
where $b/price eq 100.00
return $b/title
```

© Copyright 2003, DataDirect Technologies

# General Comparisons

- Conventional operator syntax
- Implicit existential quantification:

```
for $b in doc("books.xml")//book
where $b/author/last = "Smith"
return $b/price
```

```
for $b in doc("books.xml")//book
where some $l in $b/author/last
satisfies $l eq "Smith"
return $b/price
```

- Atomization is applied to operands
- Attempts to cast untyped data:

```
for $b in doc("books.xml")//book
where $b/price = 100.00
return $b/title
```

© Copyright 2003, DataDirect Technologies

# Untyped Data in General Comparisons

- If either operand is untyped, it is cast to a required type:
  - If the type of the other operand is numeric, the required type is xs:double.
  - If the most specific type of the other operand is xs:anySimpleType, the required type is xs:string.
  - Otherwise, the required type is the type of the other operand.
- If the cast fails, a dynamic error is raised.
- Example: 5 eq <foo>5</foo>

Colorado Software Summit

© Copyright 2003, DataDirect Technologies

# Document Order Comparisons

- Document order comparisons
  - $a << $b, $a >> $b
  - $a is $b, $a isnot $b
  - Compare two values
  - Type error if not two values
- Example

```
for $p in doc("surgery.xml")
          //section[section.title = "Procedure"]
where not(some $a in $p//anesthesia
            satisfies $a << ($p//incision)[1] )
return $p
```

© Copyright 2003, DataDirect Technologies

- Defined in XQuery syntax
- May be recursive or mutually recursive
- May have typed parameters or returns

© Copyright 2003, DataDirect Technologies

```
declare functionflatten-author( $a as element )
{
  <author>
   {
     string-value($a/last), ",", string-value($a/first)
   }
  </author>
}

for $a in doc("data/xmp-data.xml")//(author | editor)
return flatten-author($a)
```

© Copyright 2003, DataDirect Technologies

# Recursive Functions

```
declare functiondepth( $e as element ) returns integer
{
  (: An empty element has depth 1
       Otherwise, add 1 to max depth of children :)
  if (empty($e/*))
      then 1
      else max(depth($e/*)) + 1
}

depth(doc("partlist.xml"))
```

© Copyright 2003, DataDirect Technologies

# Library Modules

```
module "http://example.com/xquery/library/book"
declare functiontoc($b)
{
  for $section in $book-or-section/section
  return
    <section>
      { $section/@* , $section/title , toc($section) }
    </section>
}
```

```
import module namespace b =
"http://example.com/xquery/library/book"
      at "file:///c:/xquery/lib/book.xq"

<toc>
 {
   for $s in doc("xquery-book.xml")/book
   return b:toc($s)
 }
</toc>
```

© Copyright 2003, DataDirect Technologies

# Library Modules

- Every module is a main module or a library module.

  [30] Module ::= MainModule | LibraryModule
  [31] MainModule ::= Prolog QueryBody
  [32] LibraryModule ::= ModuleDecl Prolog

- A main module must have a query expression; a library module may not have one.
- Importing a module imports its:
  - Functions
  - Variable declarations (from the prolog)

© Copyright 2003, DataDirect Technologies

# Types in Queries

© Copyright 2003, DataDirect Technologies

# Untyped Queries

- Untyped is best for purely structural functions

```
declare functionreverse($items)
{
	let $count := count($items)
	for $i in 0 to $count
	return $items[$count - $i]
}
reverse( 1 to 5 )
```

# Predefined Types

- Predefined types require no schema
  - XML node types (element, attribute, *etc.*)
  - XML Schema built-in types

```
declare functionis-document-element($e as element())
 returns xs:boolean
{
 if ($e/.. instance of doc())
   then true()
   else false()
}
```

# Predefined Types

- An example using only values

```
declare functionfibo(n as xs:integer) as xs:integer
{
 if ($n = 0)  then 0
 else if ($n = 1)  then 1
 else (fibo($n - 1) + fibo($n - 2))
}

let $seq := 1 to 10
for $n in $seq
return <fibo n="{$n}">{ fibo($n) }</fibo>
```

© Copyright 2003, DataDirect Technologies

# Schema imports

```
import schema "urn:examples:xmp:bib" at "c:/dev/schemas/eg/bib.xsd"
default element namespace = "urn:examples:xmp:bib"

declare functionbooks-by-author($a as element(b:author))
  as element(b:title)*
{
  for $b in doc("books.xml")/bib/book
  where some $ba in $b/author satisfies
        ($ba/last=$a/last and $ba/first=$a/first)
  order by $b/title
  return $b/title
}

(: The following function call raises an error – wrong type :)

for $b in ("books.xml")/bib/book
return books-by-author($b)
```

© Copyright 2003, DataDirect Technologies

# Schema Imports

- Schemas are imported using the 'schema' expression in the prolog:

```
import schema "http://www.w3.org/1999/xhtml"
    at "http:/www.w3.org/1999/xhtml/xhtml.xsd
```

- "at" clause is optional
- Built-in XML Schema types are predefined
- Each element name, attribute name, or type name may be defined only once.
- If a schema import tries to redefine an existing name, an error results.

# Schema Import with Namespace Prefix

```
import schema namespace b = "urn:examples:xmp:bib"
  at "c:/dev/schemas/eg/bib.xsd"

declare functionbooks-by-author($a as element(b:author))
  as element(b:title)*
{
  for $b in doc("books.xml")/b:bib/b:book
  order by $b/b:title
  where some $ba in $b/b:author satisfies
         ($ba/b:last=$l and $ba/b:first=$f)
  return $b/b:title
}
```

© Copyright 2003, DataDirect Technologies

# Schema Import Is Optional

- Users need not import schemas to query typed data
- Implementations need not allow schema import
- If schemas are not imported, only predefined types can be named in a query

© Copyright 2003, DataDirect Technologies

# Function Conversion Rules

- Used to convert function parameters or returns
- Basic intuitive definition:
  - If the types match exactly, the parameter is accepted.
  - Types derived from the required type are accepted.
  - Untyped data is cast to the required type (but the cast may fail).
  - If the required type is an atomic type, and the parameter is a node, atomization is applied before trying to match the types.
- For details, see the spec

© Copyright 2003, DataDirect Technologies

- External functions are declared in the external environment (Java, SQL, *etc.*)
- Type signatures provide type safety

```
declare functionouttie($v as xs:integer)
   as xs:integer external
```

# Implicit Validation

- Element constructors with known types are implicitly validated

```
import schema "urn:examples:xmp:bib" at
            "c:/dev/schemas/eg/bib.xsd"
default element namespace = "urn:examples:xmp:bib"

<book year="1994">
  <title>Catamaran Racing from Start to Finish</title>
  <author><last>Berman</last><first>Phil</first></author>
  <publisher>W.W. Norton & Company</publisher>
</book>
```

© Copyright 2003, DataDirect Technologies

# Validating Locally Declared Elements

```
import schema namespace bib="urn:examples:xmp:bib"

validate context bib:book
 {
  <bib:price>49.99</bib:price>
 }
```

# Validation Modes

- Lax (default) validates if type is known
- Strict requires a known type, always validates

```
import schema namespace bib="urn:examples:xmp:bib"
validation strict
<bib:price>49.99</bib:price>
```

- Skip never validates

```
import schema namespace bib="urn:examples:xmp:bib"
validation strict
<bib:price>49.99</bib:price>
```

© Copyright 2003, DataDirect Technologies

# Optional Static Typing

- Error if query is inconsistent with schema
- Type errors can be detected without any data

```
import schema "urn:examples:xmp:bib" at "c:/dev/schemas/eg/bib.xsd"
default element namespace = "urn:examples:xmp:bib"
declare functionbooks-by-author($a as element(author))
  as element(title)*
{
  for $b in doc("books.xml")/bib/book
  order by $b/title
  where some $ba in $b/author satisfies
          ($ba/last=$a/last and $ba/first=$a/flirts)
  return $b/title
}
```

© Copyright 2003, DataDirect Technologies

# Predefined Types

| Sequence Type Declaration | **What it Matches** |
|---|---|
| `element()` | Any element node |
| `attribute()` | Any attribute node |
| `document-node()` | Any document node |
| node() | Any node |
| `text()` | Any text node |
| `processing-instruction()` | Any processing instruction node |
| `processing-instruction("xml-stylesheet")` | Any processing instruction node whose target is "xml-stylesheet" |
| `comment()` | Any comment node |
| `empty()` | An empty sequence |
| `item()` | Any node or atomic value |
| *QName* | An instance of a specific XML Schema built-in type, identified by the name of the type; *e.g.*, xs:string, xs:boolean, xs:decimal, xs:float, xs:double, xs:anyType… |

© Copyright 2003, DataDirect Technologies

# Occurrence Indicators

- +, ?, *
- + means "one or more":
  element()+
- ? means "zero or one":
  xs:integer?
- * means "zero or more":
  document-node()*

© Copyright 2003, DataDirect Technologies

# Types from Imported Schemas

| Sequence Type Declaration | **What it Matches** |
|---|---|
| element(creator, person) | An element named creator of type person |
| element(creator) | Any element named creator of type "xs:string" – the type declared for creator in the schema. |
| element(*, person) | Any element of type person. |
| element(book/price) | An element named price of type "currency" – the type declared for price elements inside a book element. |
| element(type(person)/la st) | An element named last of type "xs:string" – the type declared for last elements inside the person type. |
| attribute(@price, | An attribute named price of type currency. |
| attribute(book/@isbn) | An attribute named isbn of type "isbn" – the type declared for isbn attributes in a book element. |
| attribute(@*, currency) | Any attribute of type currency. |
| bib:currency | A value of the user-defined type 'currency' |

© Copyright 2003, DataDirect Technologies

# Nillable types

- The 'nillable' keyword matches nilled elements
- Example: element(n, nillable person)
  - Matches <n xsi:nil="true" />
  - Matches <n><first>Jack</first><br><last>Canada</last><br></n>

© Copyright 2003, DataDirect Technologies

# Typed Variables

- Type assertion - raise error if wrong type
- Example:

```
for $b in doc("data/xmp-data.xml")//book
let $c as element()+ := $b//author
return <count>{ count($c) }</count>
```

© Copyright 2003, DataDirect Technologies

# Variable Declarations

- Occur in query prolog
- Not to be confused with variable bindings
- Internal variable declarations:

```
declare variable $titles { doc("books.xml")//title }
```

- External variable declarations:

```
declare var $x external
declare var $i as xs:integer external
```

- Tests an item for a given type.
- Examples:
  - <foo/> instance of element()
  - 3.14 instance of xs:decimal
  - "foo" instance of xs:string
  - (1, 2, 3) instance of xs:integer*
  - () instance of xs:integer?
  - (1, 2, 3) instance of xs:integer+

© Copyright 2003, DataDirect Technologies

# typeswitch

- Chooses an expression to evaluate based on type.
- Example:

```
declare functionwrapper($x) returns element(wrap)
{
  typeswitch ($x)
      case $i as xs:integer
           return <wrap xsi:type="xs:integer">{ $i }</wrap>
      case $d as xs:decimal
           return <wrap xsi:type="xs:decimal">{ $d }</wrap>
      default
           return error("unknown type!")
}

wrapper( 1 )
```

© Copyright 2003, DataDirect Technologies

# Constructor Functions

- Used for construction or casting
- Built in schema types:

```
xs:date("2000-01-01")
```

- Constructor functions automatically generated for imported types:

```
my:currency("5.23")
```

- Facets are checked:

```
import schema namespace bib="urn:examples:xmp:bib"
bib:isbn("012345678X")
```

© Copyright 2003, DataDirect Technologies

# treat as

- Tests for correct type at run-time
- Example:

```
$myaddress treat as element(*, USAddress)
```

© Copyright 2003, DataDirect Technologies

# Summary

© Copyright 2003, DataDirect Technologies

# Why XQuery?

- Powerful transformations
- Native XML programming
- XML Views of relational data
- Optimizable in many environments
- Related to concepts people already know
- Many implementations
- The accepted W3C XML Query Language
- Preliminary update proposal

© Copyright 2003, DataDirect Technologies

# XML Query Home Page

- Pointers to all current XQuery specifications
- Over 20 XQuery implementations (at various stages)
- Articles on XQuery
- Grammar test pages
- Comments lists and discussion lists
- http://www.w3.org/XML/Query.html

© Copyright 2003, DataDirect Technologies

# Questions

- Today
- Later:
  jonathan.robie@datadirect-technologies.com
- Feedback email list:
  public-qt-comments@w3.org
- Public email list:
  www-ql@w3.org