JACOBS UNIVERSITY

Processing XML: XPath, XQuery

Ramakrishnan & Gehrke, Chapter 24 / 27

Why are we DB'ers interested?

JACOBS UNIVERSITY

- It's data, stupid. That's us.
- Database issues:
  - How are we going to model XML?
    - *Trees, graphs*
  - How are we going to query XML?
    - *XQuery*
  - How are we going to store XML?
    - *in a relational database? object-oriented? native?*
  - How are we going to process XML efficiently?
    - *many interesting research questions!*

XML Revisited

JACOBS UNIVERSITY

- From a data modelling viewpoint, what does XML offer?
- Entities (ER!)
- Attributes
  - Single-valued, atomic
- Relationships? Yes, but:
  - Single-root trees only
  - Unordered, no role names
  - General graphs through id/idrefs, syntax only

Roadmap

JACOBS UNIVERSITY

- XPath
- XQuery

Path Expressions: XPath

www.w3schools.com/xpath/

JACOBS UNIVERSITY

- Basic concept: path = sequence of location steps
  - Axis: tree relationship between nodes selected by location step + current node
    - *parent, child, self, descendant-or-self, attribute, …*
  - a node test: node type + *expanded-name* of nodes selected by location step
  - 0..* predicates: further refinement
- General location step syntax:

  axisname::nodetest[predicate]

Pattern Expressions

JACOBS UNIVERSITY

- identify nodes in document
- path through the XML document
  - .../node1/node2/...
- pattern "selects" elements that match path, result is a (sub)tree
  - „all price elements of all cd elements of the catalog element“: /catalog/cd/price

```
<price>10.90</price>
<price>9.90</price>
<price>9.90</price>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Paths

JACOBS UNIVERSITY

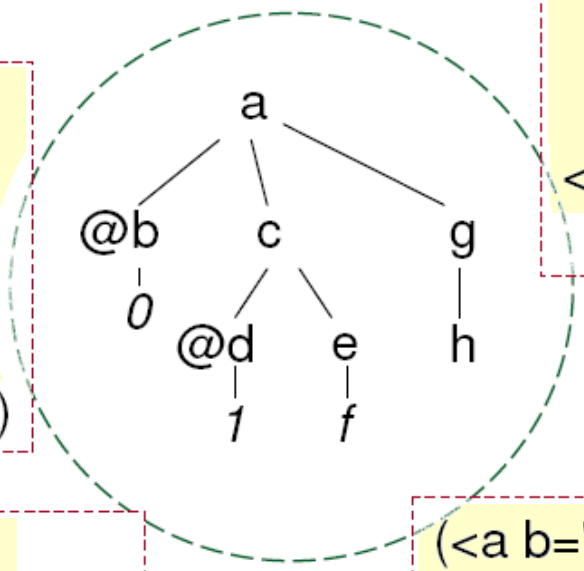
- Absolute vs. relative vs. fitting:
  - path starts with slash (/):
    absolute path
  - path starts with oduble slash (//):
    all fitting elements,
    even if at different levels in tree
  - Otherwise: path relative to current position
- Relative addressing via axis:
  - node set relative to current node
  - all children of parent, child, self, ancestor, descendant, attribute, …

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Examples

JACOBS UNIVERSITY

```
(<a b="0">
   <c d="1">
      <e>f</e>
   </c>
   <g><h/></g>
</a>)/child::node()
```




```
(<a b="0">
   <c d="1">
      <e>f</e></c>
   <g><h/></g>
</a>)/descendant::node()
```

```
(<a b="0">
      <c d="1">
         <e>f</e>
      </c>
      <g><h/></g>
</a>)/attribute::node()
```

```
(<a b="0">
      <c d="1">
         <e>f</e>
      </c>
      <g><h/></g>
</a>)/child::node()/child::node()
```

More Examples

JACOBS UNIVERSITY

```
<1>
  <2>
    <3/>
    <4/>
  </2>
  <5/>
<1/>
```

- self({2}) =
- child({1}) =
- parent({3}) =
- descendant({1}) =
- descendant-or-self({1}) =
- ancestor({4}) =
- ancestor-or-self({4}) =
- following({3}) =
- preceding({4}) =
- following-sibling({4}) =
- preceding-sibling({5}) =

Wildcards

JACOBS UNIVERSITY

- * selects unknown elements
- „*all child elements of all cd of catalog*“: /catalog/cd/*
- „*all price elements that are grandchilds of catalog*“: /catalog/*/price
- „*all price elements which have 2 ancestors*“: /*/*/price
- „*all elements*“: //*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Abbreviations

JACOBS UNIVERSITY

- a/b/c
  - ./child::a/child::b/child::c
- a//@id
  - ./child::a/descendant-or-self::node()/attribute::id
- //a
  - root(.)/descendant-or-self::node()/child::a
- a/text()
  - ./child::a/child::text()

Branch Selection

JACOBS UNIVERSITY

- Selecting branches from subtree: "[...]"
- „first cd child of catalog“: /catalog/cd[1]
  - /catalog/cd[position() = 1]
- „last cd child of catalog“: /catalog/cd[last()]
  - Note: There is no function named first()
- „*all cd elements of catalog that have a price element*“: /catalog/cd[price]
- „*all cd elements of catalog that have a price with value of 10.90*“: /catalog/cd[price=10.90]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Multiple Paths

JACOBS UNIVERSITY

- Selecting Several Paths: | operator
- „all title, artist elements“:
  /catalog/cd/title | /catalog/cd/artist
- „all the title and artist elements in the document“: //title | //artist
- „all title, artist, price elements“:
  //title | //artist | //price
- “all title elements of cd of catalog, and all artist elements“:
  /catalog/cd/title | //artist

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Attributes

JACOBS UNIVERSITY

- Selecting Attributes:
  prefix attributes with @
- „all attributes named '*country*' ":
  //@country
- „all cd elements which have an attribute named country":
  //cd[@country]
- „all cd elements with attribute named country with value 'UK' ":
  //cd[@country='UK']

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Predicates

JACOBS UNIVERSITY

- Predicates, operators, functions as usual
- „*all CDs with price below 10.0*“:
  /catalog/cd[price<10.0]
- „*all CDs with country "UK" and price below 10.0*“:
  / catalog
  / cd[@country="UK"]
  / [price<10.0]

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <cd country="USA">
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <price>10.90</price>
  </cd>
  <cd country="UK">
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <price>9.90</price>
  </cd>
  <cd country="USA">
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

Roadmap

JACOBS UNIVERSITY

- XPath
- XQuery

XQuery

JACOBS UNIVERSITY

- XQuery – retrieving information from XML data
  - XQuery = XML Query
  - XQuery is to XML
    what SQL is to tables
- extract information from XML structures
  - XPath: extract from DOM tree; XQuery: derive new structure
  - Stored in files or in database
  - Major DBMS vendors support XQuery
- See also www.w3c.org/XML/Query,
  www.w3schools.com (material borrowed)

XQuery Introductory Example

JACOBS UNIVERSITY

"*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>
```

FOR and LET

JACOBS UNIVERSITY

- FOR $x in *expr*
  - binds $x to each value in the list *expr* in turn
  - Binds *node variables* → iteration

```
FOR $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result>
    <book>...</book>
</result>
<result>
    <book>...</book>
</result>
...
```

- LET $x = *expr*
  - binds $x to the entire list *expr*
  - Defines *variable;* Binds *collection variables* → one value
  - Useful for common subexpressions and for aggregations

```
LET $x = document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result>
    <book>...</book>
    <book>...</book>
    ...
</result>
```

A More Complex Example

JACOBS UNIVERSITY

- "*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>
```

```
<result>
        <author>Jones</author>
        <title> abc </title>
        <title> def </title>
</result>
<result>
        <author> Smith </author>
        <title> ghi </title>
</result>
```

- distinct = function that eliminates duplicates

Aggregates

JACOBS UNIVERSITY

- count = (aggregate) function that returns the number of elems

```
<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>
```

```
<big_publishers>
  <publisher>Morgan Kaufmann</publisher>
  <publisher>Wiley</publisher>
</ big_publishers>
```

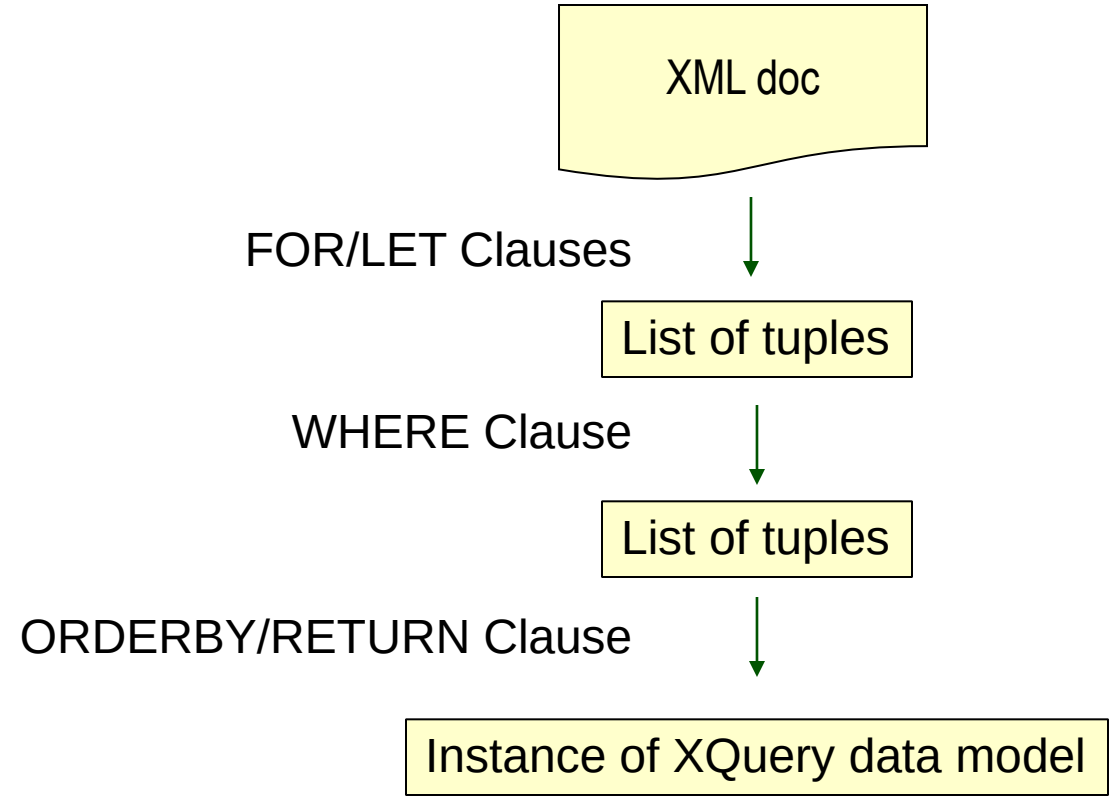
- How to obtain that?

```
<num_big_publishers>120</ num_big_publishers>
```

Summary: General Query Structure

JACOBS UNIVERSITY

- FOR-LET-WHERE-ORDERBY-RETURN
  = FLWOR ("*flower*")
- XPath 2.0 supports FLOWR as well!
  - But not further "advanced" stuff of XQuery



Summary: XML Family (Excerpt)

JACOBS UNIVERSITY