Chapter 4

XML Query Languages

Foundations
XML Path Language (XPath) 2.0
XQuery 1.0: An XML Query Language
XIRQL
History

1998
- XSLT
- XQL
- XML-QL
- UnQL
- Lorel

1999
- XPath 1.0
- Quilt
- XML Schema

2000

2001

2003
- SQL/XML

2007
- XPath 2.0
- XQuery 1.0

- SQL
- OQL
- W3C Recommendations
- Other Proposals
- Standard DB Query Languages

Lecture "XML and Databases" - Dr. Can Türker
Basic Query Language Requirements

- **Ad-hoc**: Formulate queries without writing complete programs
- **Declarative**: Describe what is searched, not how the search should be computed
- **Generic**: Query language is built upon a few generic operations
- **Set-Oriented**: Operations work on set of objects
- **Adequate**: All constructs of the data model are exploited
- **Orthogonal**: All operations can be combined
- **Closed**: Query results can used as input for other queries
- **Complete**: All stored informations can be retrieved
- **Optimizable**: Queries can be optimized using equivalence rules
- **Efficient**: Operations can be implemented efficiently
- **Safe**: Queries always terminate and deliver a finite result
- **Formal Semantics**: All operations are formally defined
Relational Model and Algebra

- Information represented by relations (tables) \( \approx \) sets of tuples (rows)
- All attributes of tuples are atomic
- Algebra operations:
  - Selection: select tuples (rows)
  - Projection: select attributes (columns)
  - Set operations: relation union, except (difference), intersect
  - Join: combine tuples / relations
  - Rename: rename attributes (columns)
- All operations can be combined
- Relational algebra provides fundament for query optimization

- Hint: Combined algebra expressions are read from right to left!
Relational Algebra Operations

**Selection** \( \sigma[A1<3](R1) \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Jim'</td>
</tr>
<tr>
<td>2</td>
<td>'Dad'</td>
</tr>
</tbody>
</table>

**Projection** \( \pi[A1](R1) \)

<table>
<thead>
<tr>
<th>A1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

**Rename** \( \beta[A1,X1](R1) \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Jim'</td>
</tr>
<tr>
<td>2</td>
<td>'Dad'</td>
</tr>
<tr>
<td>3</td>
<td>'Joe'</td>
</tr>
</tbody>
</table>

**Union** \( R1 \cup R2 \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Jim'</td>
</tr>
<tr>
<td>2</td>
<td>'Dad'</td>
</tr>
<tr>
<td>2</td>
<td>'Dad'</td>
</tr>
<tr>
<td>3</td>
<td>'Joe'</td>
</tr>
<tr>
<td>3</td>
<td>'Bob'</td>
</tr>
</tbody>
</table>

**Except** \( R1 \setminus R2 \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'Jim'</td>
</tr>
<tr>
<td>3</td>
<td>'Joe'</td>
</tr>
</tbody>
</table>

**Intersect** \( R1 \cap R2 \)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>'Dad'</td>
</tr>
</tbody>
</table>

**Join** \( R1 \times R3 \)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>'Dad'</td>
<td>'F'</td>
</tr>
<tr>
<td>3</td>
<td>'Joe'</td>
<td>'T'</td>
</tr>
</tbody>
</table>
Nested Relations

- **NF2 (Non First Normal Form)** model supports atomic and relation-valued attributes
- **Minimal extension of relational algebra** includes operations for relation-valued attributes
  - Access to inner structures of nested relations via recursive nesting of selections and projections within selection predicates and projection lists
  - Selection predicates can include relational conditions
    - Set comparisons ($\subseteq$, $\subset$, $=$, $\neq$, $\supset$, $\supseteq$)
    - Set inclusion ($\in$, $\notin$)
  - Nesting / Unnesting

\[
\begin{array}{ccc}
R & A1 & A2 & A3 \\
2 & 'Kim' & 'T' & \ \\
2 & 'Dad' & 'F' & \ \\
3 & 'Joe' & 'T' & \\
\end{array}
\]

\[
\begin{array}{ccc}
R' & A1 & \{A23\} \\
A2 & A3 \\
2 & 'Kim' & 'T' \\
2 & 'Dad' & 'F' \\
3 & 'Joe' & 'T' \\
\end{array}
\]

\[
\nu[(A2, A3); A23] (R) \quad \mu[A23] (R')
\]
Nesting Selections and Projections

\[ \pi[\pi[A2](A23)](R) \]
\[ \{R' \} \]
\[ \{A23 \} \]
\[ A2 \]
\[ 'Kim' \]
\[ 'Dad' \]
\[ 'Joe' \]

\[ \pi[\sigma[A3='T'](A23)](R) \]
\[ \{R' \} \]
\[ \{A23 \} \]
\[ A2 \]
\[ A3 \]
\[ 'Kim' \]
\[ 'T' \]
\[ 'Joe' \]
\[ 'T' \]

\[ \sigma[\emptyset \cap \pi[A2](A23)](R) \]
\[ \{R' \} \]
\[ A1 \]
\[ \{A23 \} \]
\[ A2 \]
\[ A3 \]
\[ 2 \]
\[ 'Kim' \]
\[ 'T' \]
\[ 'Dad' \]
\[ 'F' \]
\[ 3 \]
\[ 'Joe' \]
\[ 'T' \]

\[ \sigma[\emptyset \cap \sigma[A3='F'](A23)](R) \]
\[ \{R' \} \]
\[ A1 \]
\[ \{A23 \} \]
\[ A2 \]
\[ A3 \]
\[ 2 \]
\[ 'Kim' \]
\[ 'T' \]
\[ 'Dad' \]
\[ 'F' \]
\[ 3 \]
\[ 'Joe' \]
\[ 'T' \]
Extended Relational Models (1)

- Support for further attribute types: tuple type, collection types, reference type

- Operations on tuples
  - Tuple field access
  - Navigation: Access fields of nested tuples

- Operations on collections
  - Element containment and subset associations
  - Unnesting

\[ \sigma[A2.A21>1971](R) \]

\[
\begin{array}{ccc}
A1 & <A2>  \\
A21 & \{A22\}  \\
1 & 1970 & \{41, 16\}  \\
2 & 1972 & \{12, 1, 78\}  \\
3 & 1969 & \{13, 11, 69\}  \\
\end{array}
\]

\[ \sigma[12 \in A2.A22](R) \]

\[
\begin{array}{ccc}
A1 & <A2>  \\
A21 & \{A22\}  \\
2 & 1972 & \{12, 1, 78\}  \\
\end{array}
\]
Extended Relational Models (2)

- Operations on references
  - Dereferencing: Access the referenced object
  - Navigation: Access attributes of referenced objects using path expressions

<table>
<thead>
<tr>
<th>R</th>
<th>TID</th>
<th>Name</th>
<th>Boss</th>
</tr>
</thead>
<tbody>
<tr>
<td>@911</td>
<td>'Jim'</td>
<td>@655</td>
<td></td>
</tr>
<tr>
<td>@655</td>
<td>'Joe'</td>
<td>@324</td>
<td></td>
</tr>
<tr>
<td>@876</td>
<td>'Bob'</td>
<td>@655</td>
<td></td>
</tr>
<tr>
<td>@324</td>
<td>'Kim'</td>
<td>@324</td>
<td></td>
</tr>
</tbody>
</table>

\[ \pi[\text{Name, } \text{DEREF(Boss)}](R) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>&lt;\text{DEREF(Boss)}&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Jim'</td>
<td>'Joe' @324</td>
</tr>
<tr>
<td>'Joe'</td>
<td>'Kim' @324</td>
</tr>
<tr>
<td>'Bob'</td>
<td>'Joe' @324</td>
</tr>
<tr>
<td>'Kim'</td>
<td>'Kim' @324</td>
</tr>
</tbody>
</table>

\[ \pi[\text{Name, Boss }\to\text{Name}](R) \]

<table>
<thead>
<tr>
<th>Name</th>
<th>Boss $\to$ Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Jim'</td>
<td>'Joe'</td>
</tr>
<tr>
<td>'Joe'</td>
<td>'Kim'</td>
</tr>
<tr>
<td>'Bob'</td>
<td>'Joe'</td>
</tr>
<tr>
<td>'Kim'</td>
<td>'Kim'</td>
</tr>
</tbody>
</table>

- Hint: The arrow symbol is a shortcut for a dereferencing (\text{DEREF}) followed by a tuple field access (\text{.}). Sometimes, the dot symbol is used instead of the arrow symbol. This however might cause confusion since the dereferencing is then done implicitly.
What can be done so far?

- Selection on sets of tuples/objects
- Projection on certain attributes
- Join sets of tuples/objects
- Classic set operations on tuples/objects
- Navigation in nested structures: path expressions on embedded tuples/objects
- Navigation in network structures: path expressions using references
- ...

What is missing for XML Query Languages?
Additional Requirements for XML Query Languages

- **Schema Awareness**
  - Queries must be possible whether or not a schema is available; Support “wildcards“ in path expressions

- **Flexible Types**
  - Processing elements of different types

- **Seamless XML Embedding**
  - Queries embedded with XML and XML embedded in Queries

- **Ordering**
  - Element order must be retained
Basic Operations of XML Query Languages

- Selection
  - select documents or elements based on content and structure as well

- Extraction and Reduction
  - extract and delete sub elements

- Combination and Restructuring
  - compose two or more elements combined to a new element, create new element sequences
Selection

\[ \sigma[//C/E/F] \text{ } $document() \]

selection predicate based on the document structure
Extraction and Reduction

\[ \pi[//C] \text{ $document()$} \]

Lecture "XML and Databases" - Dr. Can Türker
Combination and Restructuring

Element constructor $\varepsilon[X]$ creates a new element

«B elements are renamed to Y, then C and Y elements are placed as sub elements of a new element X»
XML Path Language (XPath) 2.0

- W3C Recommendation 23 January 2007

- XPath defines pattern, functions, and expressions to select XML elements and attributes
  - Addressing node sets
  - Formulating conditions on these node sets

- Basic construct: XPath expressions
  - Expressions are of type boolean, number, string, or node-set (unordered collection of nodes)
  - Path expressions
  - Logical and mathematical operators
  - Function calls

- XPath is part of several standards
  - XSL, XLink/XPointer, XQuery
Data Model of XPath/XQuery

- **Bases on XML Information Set**
  - XML 1.0, Namespaces, XML Schema

- **Data types**
  - Simple and complex types known from XML Schema
  - XML 1.0 Characters
  - XPath node types (document, element, attribute, text, namespace, comment, processing instruction)

- **With and without schema**

- **XPath/XQuery expressions return a sequence of items**
  - Ordered collection of null or more *items*
  - An *item* is a *atomic value* or a *node*
  - Atomic value (of a XML Schema type): `string` | `boolean` | `decimal` | `ID` | `IDREF` ...
  - Node: `document` | `element` | `attribute` | `text` | `namespaces` | `comment` | `PI`
  - Values and nodes can be typed or untyped
**Built-in Data Types**

### Predefined Namespaces - Prefixes

- xs: http://www.w3.org/2001/XMLSchema
- xsi: http://www.w3.org/2001/XMLSchema-instance
- fn: http://www.w3.org/2003/11/xpath-functions
- xdt: http://www.w3.org/2003/11/xpath-datatypes
- local: http://www.w3.org/2005/xquery-local-functions

* Figure taken from XQuery 1.0 and XPath 2.0 Data Model (XDM) W3C Recommendation 23 January 2007

http://www.w3.org/TR/xpath-datamodell/
Main Data Type: Sequence

- A sequence of one item equals the same item
  - $(1) \equiv 1$

- Sequences are implicitly unnested
  - $(1, (2, 3)) \equiv (1, 2, 3)$

- Sequences can be heterogeneous
  - $(<a/>, 3)$

- Sequences can contain duplicates
  - $(2, 2, 2)$
Path Expressions

- Address nodes of an XML tree
- Designed to be embedded in a host language
- Have the form: `/Step/Step/…/Step`
  - can consist of several expressions (steps) that are connected via the slash (/) symbol
- Stepwise processing from left to right

```xml
//book[title='XML and Databases']/author
```

yields all author elements of book elements whose title element has the content 'XML and Databases'

- Absolute versus relative path expressions
- Note: Path expressions define extractions and also selections by using filter predicates
Steps

- Input and output of a step is a sequence

- **Step expression:** `NavigationAxis::NodeTest[Predicate]`
  - `NavigationAxis` defines relationship between context node and nodes to be selected
  - `NodeTest` includes type and name of nodes to be selected
  - `Predicate` provides filter for nodes to be selected

- **Steps processed within a context**

- **Processing context includes**
  - context node (`self`)
  - context position and size
  - set of namespace declarations
Navigation Axis

- **ancestor**
  - \( = \text{ancestor} \cup \text{self} \)

- **descendant**

- **child**

- **preceding**

- **sibling**

- **parent**

- **self**

- **following**

- **ancestor-or-self**
  - \( = \text{ancestor} \cup \text{self} \)

- **descendant-or-self**
  - \( = \text{descendant} \cup \text{self} \)

- **attribute**

- **namespace**
**Node Test**

- **Restricts types and names of the nodes to be selected**

<table>
<thead>
<tr>
<th>Node Test</th>
<th>Restriction to</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>no restriction (&quot;wildcard&quot;)</td>
</tr>
<tr>
<td>name</td>
<td>all sub elements with the given name</td>
</tr>
<tr>
<td>document-node()</td>
<td>the document node</td>
</tr>
<tr>
<td>node()</td>
<td>all sub element nodes</td>
</tr>
<tr>
<td>text()</td>
<td>all text element nodes</td>
</tr>
<tr>
<td>processing-instruction()</td>
<td>all processing instruction nodes</td>
</tr>
<tr>
<td>comment()</td>
<td>all comment nodes</td>
</tr>
</tbody>
</table>
Predicate

- **Filter expression on a node sequence**

- **Combined predicates**
  - Example: `/book[3]/author/last-name='Melville'`
  - Evaluation from left to right
  - NOT commutative: \(a[b][2] \neq a[2][b]\)

- **Conjunctive predicates**
  - Example: `/book[price<15 \land count(author)>2]`

- If a predicate does not yield a boolean value, it is implicitly converted to a boolean value
  - Numeric values converted to position predicates
  - In all other cases the result of the conversion is false
# Syntax: Shortcuts

<table>
<thead>
<tr>
<th>Shortcut</th>
<th>Full Expression</th>
<th>yields</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagname</td>
<td>child::tagname</td>
<td>all child nodes referring to tagname elements</td>
</tr>
<tr>
<td>.</td>
<td>self::node()</td>
<td>the current context node</td>
</tr>
<tr>
<td>..</td>
<td>parent::node()</td>
<td>the parent node</td>
</tr>
<tr>
<td>*</td>
<td>descendant-or-self::</td>
<td>all descendant nodes</td>
</tr>
<tr>
<td>@name</td>
<td>attribute::name</td>
<td>attribute 'name' of the current node</td>
</tr>
<tr>
<td>/</td>
<td></td>
<td>the root node</td>
</tr>
<tr>
<td>//</td>
<td></td>
<td>all descendant nodes of the root</td>
</tr>
<tr>
<td>[expr]</td>
<td></td>
<td>elements of node sequence that satisfy the expression</td>
</tr>
<tr>
<td>[n]</td>
<td></td>
<td>n-th element of a node sequence</td>
</tr>
</tbody>
</table>
## XPath Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>yields</th>
</tr>
</thead>
<tbody>
<tr>
<td>number last()</td>
<td>position of the last element</td>
</tr>
<tr>
<td>number position()</td>
<td>context position</td>
</tr>
<tr>
<td>number sum(node-set)</td>
<td>sum of the node values</td>
</tr>
<tr>
<td>number count(node-set)</td>
<td>count of the nodes</td>
</tr>
<tr>
<td>string name(node-set?)</td>
<td>name of the node set</td>
</tr>
<tr>
<td>node-set id(object)</td>
<td>nodes with this id</td>
</tr>
<tr>
<td>boolean contains(string, string)</td>
<td>true if second argument is contained in the first argument</td>
</tr>
<tr>
<td>boolean not(boolean)</td>
<td>negation</td>
</tr>
</tbody>
</table>

- **Further functions:**
  - number, floor, ceiling, round
  - string, concat, starts-with, ends-with, substring-before, substring-after, substring, string-length, normalize-space, translate
  - base-uri, document-uri, namespace-uri, node-name

- **Document access with** `fn:doc(uri)`
XPath-Queries Examples

bs.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book genre="autobiography">
    <title>The Autobiography of Benjamin Franklin</title>
    <author>
      <first-name>Benjamin</first-name>
      <last-name>Franklin</last-name>
    </author>
    <price currency="USD">8.99</price>
  </book>
  <book genre="novel">
    <title>The Confidence Man</title>
    <author>
      <first-name>Herman</first-name>
      <last-name>Melville</last-name>
    </author>
    <price currency="USD">11.99</price>
  </book>
  <book genre="philosophy">
    <title>The Gorgias</title>
    <author>
      <name>Plato</name>
    </author>
    <price currency="USD">9.99</price>
  </book>
</bookstore>
```

fn:doc("bs.xml")/bookstore/book/@genre

```
genre="autobiography"
genre="novel"
genre="philosophy"
```

fn:doc("bs.xml")/bookstore/book[author/name='Plato']

```
<book genre="philosophy">
  <title>The Gorgias</title>
  <author><name>Plato</name></author>
  <price currency="USD">9.99</price>
</book>
```

fn:doc("bs.xml")//author[first-name='Herman']/last-name

```
<std last-name>Melville</std>
```

fn:doc("bs.xml")//book[author/last-name='Franklin']/price

```
<price currency="USD">8.99</price>
```

fn:doc("bs.xml")//book[contains(title, 'an')]/title

```
<title>The Autobiography of Benjamin Franklin</title>
<title>The Confidence Man</title>
```

fn:doc("bs.xml")//book[.//name='Plato' and price < 20]/title

```
<title>The Gorgias</title>
```
XPath
Processing Model

* figure taken from

XML Path Language (XPath) 2.0
W3C Recommendation 23 January 2007

http://www.w3.org/TR/xpath20/
Conclusions: XPath

- Tree-based data model
- Queries formulated as path expressions
- Well-defined semantics
- XPath supports
  - Extraction and reduction (described by steps of path expressions)
  - Selection (described by filter predicates in steps)
  - Aggregate functions (count, sum)
  - Navigation functions
  - Wildcards
  - Order preservation

- No support for combination and restructuring!
XQuery 1.0: An XML Query Language

- W3C Recommendation 23 January 2007
- Based on the XPath/XQuery data model
- Strongly typed based on XML Schema
- Similar to SQL/OQL
- Functional language but also includes imperative constructs
- Supports composite expressions and orthogonal usage of different expression types

→ XQuery is more than a declarative query language
→ Programming language for arbitrary XML transformations
XQuery Basics

- Embedding XML in XQuery expressions and vice versa
- Element constructors and computed XML elements
- Path expressions (XPath 2.0) for selection of node sequences
- Data type specific operators
- FLWOR expressions allow queries similar to SFW clauses in SQL
  - for/let: ordered list of tuples of bound variables
  - where: restricted list of tuples of bound variables
  - order: sorted list of tuples of bound variables
  - return: result construction which is an instance of the XQuery data model
- Conditional statements
- Quantified expressions using the ALL and SOME quantifiers
- Data type testing and conversion
- Function calls
XQuery Prolog

- Every query in XQuery consists of an expression and an optional prolog which defines the context for the expression evaluation

- Prolog can contain different types of declarations:
  - XQuery version
  ```xquery version "1.0" encoding "utf-8";
  ```
  - Global and external variables
    ```define variable $x external;
define variable $copyright as xs:string := "Copyright 2003-2007";
  ```
  - Document order
    ```declare ordering ordered;
declare ordering unordered;
  ```
  - Functions
    ```declare function local:depth($e as node()) as xs:integer {
        if (fn:empty($e/*)) then 1 else fn:max(for $c in $e/*
        return local:depth($c)) + 1
    };
  ```
  - Namespaces
  - Import of schemata and function libraries
  - ...
## XQuery Expressions

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructors</td>
<td>pcdata(expr), processing-instruction(expr, expr), comment(expr), etc.</td>
</tr>
<tr>
<td>Navigation methods</td>
<td>children(expr), parent(expr), attributes(expr), name(expr), etc.</td>
</tr>
<tr>
<td>Arithmetic functions</td>
<td>+</td>
</tr>
<tr>
<td>Comparison functions</td>
<td>=</td>
</tr>
<tr>
<td>Aggregate functions</td>
<td>agg(expr) with agg ∈ {count, min, max, sum, avg}</td>
</tr>
<tr>
<td>Set functions</td>
<td>union</td>
</tr>
<tr>
<td>Iterator</td>
<td>for variable in expr return expr</td>
</tr>
<tr>
<td>Conditions</td>
<td>if (expr) then expr else expr</td>
</tr>
<tr>
<td>Local variable binding</td>
<td>let variable := expr</td>
</tr>
<tr>
<td>Sorting</td>
<td>expr order by (expr)</td>
</tr>
<tr>
<td>Document access</td>
<td>fn:doc(uri) or fn:collection(uri)</td>
</tr>
</tbody>
</table>
XQuery FLWOR Expressions

```
flwor-exp ::= (for-exp | let-exp)+
    (where expr)?
    (order by expr)?
return expr

for-exp ::= (for $var in expr (, $var in expr)*)+

let-exp ::= (let $var := expr (, $var := expr)*)+
```

```
for $v1 in e1, $v2 in e2, ..., $vn in en
where SelectionPredicate
order by OrderExpression
return ProjectionList
```

```
SELECT ProjectionList
FROM e1 $v1, e2 $v2, ..., en $vn
WHERE SelectionPredicate
ORDER BY OrderExpression
```

```
for $v1 in e1 for $v2 in $v1
where SelectionPredicate
order by OrderExpression
return ProjectionList
```

```
SELECT ProjectionList
FROM e1 $v1, UNNEST($e2) $v2
WHERE SelectionPredicate
ORDER BY OrderExpression
```
XQuery Variables

- Binding in for and let expressions
- Type derived from the binding
- Values fixed with binding
- Binding visible only within the current and all included query expressions
- Binding released with finishing the expression evaluation
- In case of several bindings, the last one is visible
Atomization

- **The fn:data function** accepts a sequence of items and returns their typed values
  - For atomic values: return the value itself
  - For nodes: extract the typed value of the node

- **Calling fn:data is often unnecessary** because the typed value of a node is automatically extracted (atomized) for many XQuery/XPath expressions, including comparisons, arithmetic operations, function calls

```xml
<result>
  <f1>{fn:data(fn:doc("bookstore.xml")//book)}</f1>
  <f2>{fn:data(fn:doc("bookstore.xml")//@genre)}</f2>
  <f3>{fn:data(fn:doc("bookstore.xml")//book[1]/title)}</f3>
  <f4>{fn:data(fn:doc("bookstore.xml")//book[1]/title/text())}</f4>
</result>
```

yields

```xml
<result>
  <f1>The Autobiography of Benjamin Franklin Benjamin Franklin 8.99
  The Confidence Man Herman Melville 11.99
  The Gorgias Plato Plato Plato 9.99</f1>
  <f2>autobiography novel philosophy</f2>
  <f3>The Autobiography of Benjamin Franklin</f3>
  <f4>The Autobiography of Benjamin Franklin</f4>
</result>
```
Example Data and Schema

```xml
type Bib = element bib (Book*)
type Book = element book
  (attribute year (xs:integer) &
   attribute isbn (xs:string),
   element title (xs:string),
   (element author(xs:string))+)

let $bib0 :=
  <bib>
  <book year="1999" isbn="1-55860-622-X">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
  <book year="2001" isbn="1-XXXXX-YYY-Z">
    <title>XML Query</title>
    <author>Fernandez</author>
    <author>Suciu</author>
  </book>
</bib>
return $bib0
```

```xml
let $book0 :=
  <book year="1999" isbn="1-55860-622-X">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
return $book0
```
Element and Attribute Constructor

- Element construction using the XML notation

  `<lecture>XML and Databases</lecture>`

- XQuery expressions are wrapped by curly brackets `{ }`

- The curly brackets `{ and } are masked by doubling

  `<references lecture="XML and Databases">
  { for $x in fn:$bib0//@book
    return <book title={$x/title/text()}>{string($x/@ISBN)}</book>
  }
  </references>`

  yields

  `<references lecture="XML and Databases">
  </references>`
Extraction and Reduction

(: projection on elements:)
$bib0/bib/book/author$

yields

<author>Abiteboul</author>
<author>Buneman</author>
<author>Suciu</author>
<author>Fernandez</author>
<author>Suciu</author>

(: projection on element content :)
for $a$ in $bib0/bib/book/author$
return &lt;a&gt;{fn:data($a)}&lt;/a&gt;

yields

<a>Abiteboul</a>
<a>Buneman</a>
<a>Suciu</a>
<a>Fernandez</a>
<a>Suciu</a>

(: projection on attribute :)
<y>{$book0/book/@year}</y>

yields

<y year="1999"/>

(: projection on attribute values :)
<y>{fn:data($book0/book/@year)}</y>

yields

<y>1999</y>
Iteration

(() iteration over elements :)

For $b$ in $\text{bib0/bib/book}$
Return `<book><typeof b, $b/title</book>`

Yields

<book>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <author>Suciu</author>
  <title>Data on the Web</title>
</book>
<book>
  <author>Fernandez</author>
  <author>Suciu</author>
  <title>XML Query</title>
</book>

Element construction using pure XML or composite expressions (as here!)

For $b$ in $\text{bib0/bib/book}$
Return $b/author$

Is equivalent to

$\text{bib0/bib/book/author}$
Selection

(: selection of elements :)

```xml
for $b$ in $\text{bib0/bib/book}$
where $b/@year \leq 2000$
return $b$
```

yields

```xml
<book year="1999" isbn="1-55860-622-X">
<title>Data on the Web</title>
<author>Abiteboul</author>
<author>Buneman</author>
<author>Suciu</author>
</book>
```

Predicate can be a complex one consisting of several parts

```xml
for $b$ in $\text{bib0/bib/book}$
where true
return $b/author$
```

is equivalent to

```
$\text{bib0/bib/book/author}$
```
Quantification

( : using existence quantifier : )

for $b$ in $\bib0/bib/book$
where some $a$ in $b/author$
  satisfies $a = "Buneman"
return $b$

yields

<book year="1999" isbn="1-55860-622-X">
  <title>Data on the Web</title>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <author>Suciu</author>
</book>

( : using all quantifier : )

for $b$ in $\bib0/bib/book$
where every $a$ in $b/author$
  satisfies $a = "Buneman"
return $b$

yields

()
**Combination and Restructuring**

```xml
<reviews>
  <book>
    <title>Data on the Web</title>
    <review>A darn fine book.</review>
  </book>
  <book>
    <title>XML Query</title>
    <review>This is great!</review>
  </book>
</reviews>
```

```xml
let $review0 :=
<reviews>
  <book>
    <title>Data on the Web</title>
    <review>A darn fine book.</review>
  </book>
  <book>
    <title>XML Query</title>
    <review>This is great!</review>
  </book>
</reviews>
```

```xml
for $b in $bib0/bib/book,
  $r in $review0/reviews/book
where $b/title = $r/title
return
  <book> {$b/title, $b/author, $r/review} </book>
```

Yields

```xml
<book>
  <title>Data on the Web</title>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <author>Suciu</author>
  <review>A darn fine book.</review>
</book>
<book>
  <title>XML Query</title>
  <author>Fernandez</author>
  <author>Suciu</author>
  <review>This is great!</review>
</book>
```

Join condition
Sorting

```xml
for $b in $review0//book
order by $b/title ascending
return $b
```

yields

```xml
<book>
  <title>Data on the Web</title>
  <review>A darn fine book.</review>
</book>
<book>
  <title>XML Query</title>
  <review>This is great!</review>
</book>
```
for $a$ in distinct-values($bib0//author$)
let $b := $bib0//book[author=$a$
return <group>
 {
 $a$
 {<count>${count}($b)</count>}</group>

yields

<group>
 <author>Abiteboul</author><count>1</count>
</group>
<group>
 <author>Buneman</author><count>1</count>
</group>
<group>
 <author>Suciu</author><count>2</count>
</group>
<group>
 <author>Fernandez</author><count>1</count>
</group>
for $b$ in $bib0/bib/book$
where $b/@year = 2001$
return $b/..$

yields

<bib>
  <book year="1999" isbn="1-55860-622-X">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
  </book>
  <book year="2001" isbn="1-XXXXX-YYY-Z">
    <title>XML Query</title>
    <author>Fernandez</author>
    <author>Suciu</author>
  </book>
</bib>
Type Conversion: Treat and Cast

```
for $p in $book0/book/
return $p treat as Book
```

yields

```
<book year="1999" isbn="1-55860-622-X">
<title>Data on the Web</title>
<author>Abiteboul</author>
<author>Buneman</author>
<author>Suciu</author>
</book>
```

Declared node type=Book

**Type cast**
- `treat` → static
- `cast` → dynamic

Semantics differs from TREAT and CAST in SQL:1999

```
type Book0 = element book
  (attribute year (xs:integer) &
   attribute isbn (xs:string),
   element title (xs:string),
   (element author (xs:string))*)

for $p in $book0/book/
return $p cast as Book0
```

yields

```
<book year="1999" isbn="1-55860-622-X">
<title>Data on the Web</title>
<author>Abiteboul</author>
<author>Buneman</author>
<author>Suciu</author>
</book>
```

Most special node type=Book0
XQuery Processing Model

* figure taken from XQuery 1.0: An XML Query Language (Second Edition) W3C Recommendation 14 December 2010

http://www.w3.org/TR/xquery/
### Comparison XQuery and SQL

<table>
<thead>
<tr>
<th>XQuery</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>for $k in /bookstore/book return $k</td>
<td>SELECT * FROM bookstore</td>
</tr>
<tr>
<td>for $k in //book return $k</td>
<td>SELECT * FROM bookstore</td>
</tr>
<tr>
<td>for $k in //book/title return $k</td>
<td>SELECT title FROM bookstore</td>
</tr>
<tr>
<td>for $k in //book/title return $k</td>
<td>SELECT title FROM bookstore</td>
</tr>
<tr>
<td>for $k in //book return $k/title</td>
<td>SELECT title FROM bookstore</td>
</tr>
<tr>
<td>for $k in //book/author return $k/last-name</td>
<td>SELECT author.last-name FROM bookstore</td>
</tr>
<tr>
<td>for $k in /bookstore/book where $k/title='XML and Databases' order by $k/author/last-name return $k/author</td>
<td>SELECT author FROM bookstore ORDER BY author.last-name WHERE title='XML and Databases'</td>
</tr>
<tr>
<td>for $k in /bookstore/book where count($k/author) &gt; 2 return $k/title</td>
<td>SELECT title FROM bookstore GROUP BY title HAVING COUNT(author) &gt; 2</td>
</tr>
</tbody>
</table>
Summary: XQuery

- **Standard for XML Query Languages**
- **Based on a tree model that supports all XML node types**
- **Well-defined semantics**
- **Strongly typed**
- **Support all requirements of XML Query Languages**
  - Selection
  - Extraction and reduction
  - Combination and restructuring
  - Preservation document order
- **Provides SQL goodies**
  - Grouping and aggregate functions
  - Sorting
  - Dynamic and static typing
Extending Query Languages with IR Functionality

- **XPath/XQuery focus on largely structured XML documents**
  - Precise predicates (exact match)
  - Well-defined result sets
  - Operations: selection, extraction, restructuring, aggregation
  → Data-oriented view

- **Document-oriented View**
  - XML used as format for representing the logical structure of (text) documents
  - XPath/XQuery support only simple boolean retrieval but NOT IR on XML documents
    - No search for single word occurrences and substring matches
    - No weighting of descriptors
    - No relevance-oriented ranking of result sets

- **IR extensions needed**
  - Weighting and ranking
  - Relevance-oriented search
  - Data types with vague predicates
  - Structural relativism
Weighting and Ranking

- Classic IR considers only entire documents
- XML Retrieval in contrast can restrict conditions to specific parts of the documents
  - `/document[.//heading ↭ "XML" ∨ .//section//* ↭ "XML"]`
- Problem: Weighting terms of different types
Relevance-Oriented Search

- **Content-only Queries**
  - Expressions do not refer to the document structure
  - Example: "Search for XML Query Languages"
  - Retrieval strategy: return most specific sub tree that matches the given query best, i.e., has the highest retrieval status value

- **Content-and-Structure Queries**
  - Expressions formulate restrictions on the document structure
  - Example: "Search for all abstract or conclusion elements dealing with XML Query Languages"
  - Retrieval strategy: return the structure elements with the highest retrieval status value which satisfy the conditions on the structure
Data Types with Vague Predicates

- **Example**
  - Query: Search for informations about the work of an artist called Ulbrich who was active around 1900 in the Rhein/Main area
  - Actual target: Ernst Olbrich, Darmstadt, 1899

- **Extended data types for document-oriented view**
  - Person names
  - Dates
  - Geographic nomenclature
  - Images, audio, video, ...

- **Idea**: Exploit XML markups for formulating more precise search queries while considering uncertainty and vagueness
Structural Relativism

- **XPath only supports precise conditions in path expressions**
  - Example: `/store/auction/name[last-name="Schek"]`
  - Example with wildcards: `//name[last-name="Schek"]`
  - Query writing requires good knowledge about the structure of the given documents
  - In big document collections, it is unrealistic that a user has this knowledge

- **Structural relativism extends relevance-oriented search to paths and path expressions**
  - No distinction between elements and attributes
  - Search in all elements of a given data type, e.g. Date
  - Search for elements that contain a given keyword in their path
  - Search for elements that are on a path with the highest relevance w.r.t. a given query text
Query Language XIRQL

- **Extension of XPath expressions**
  - Probabilistic retrieval based on weighted query conditions
    - `//*[0.7 . $c-word$ "retrieval" + 0.3 . $c-word$ "XML"]`
  - Relevance-oriented search: IR search restricted to selected parts of XML documents
    - `//section[... $c-phrase$ "XML retrieval"]`
  - Data types with vague predicates instead of "=" or "<"
    - Keyword search: `//title $c-word$ "autobiography"`
    - Phonetic match: `//author $soundslike$ "franklin"`
  - Structural relativism do not distinguish between elements and attributes
    - `//#author $soundslike$ "franklin"`

- **XIRQL provides a set of operators and allows to define own new operators and data types**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeset $c-word$</td>
<td>String Weighted search for word occurrences</td>
</tr>
<tr>
<td>nodeset $c-phrase$</td>
<td>String Weighted search for phrase occurrences</td>
</tr>
<tr>
<td>nodeset $soundslike$</td>
<td>String Weighted phonetic search</td>
</tr>
<tr>
<td>#name</td>
<td>String No distinction between attributes and elements</td>
</tr>
</tbody>
</table>
XIRQL: Example Document

<bib>
  <book year="1999" isbn="1-55860-622-X">
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <abstract>The Web is causing a revolution in how we present, retrieve, and process information. ...</abstract>
  </book>
  <book year="2001" isbn="1-XXXXX-YYY-Z">
    <title>XML Query</title>
    <author>Fernandez</author>
    <author>Suciu</author>
    <summary>...</summary>
  </book>
</bib>
XIRQL: Example Query (1)

**Weighted Keyword Search**

```xml
//book[abstract $c-word$ "Web"]/title
```

yields

```xml
<title rsv="0.75">Data on the Web</title>
<title rsv="0.1">XML Query</title>
```

Weighting the results
**XIRQL: Example Query (2)**

**Individual Weighting of Query Conditions**

```
//book[0.7/.abstract $c-word$ "Web" + 
0.3/.author $soundslike$ "Sutschu"]
```

yields

```
<book year="1999" isbn="1-55860-622-X" rsv="0.6">
<title>Data on the Web</title>
<author>Abiteboul</author>
<author>Buneman</author>
<author>Suciu</author>
<abstract>...</abstract>
</book>

<book year="2001" isbn="1-XXXXX-YYY-Z" rsv="0.2">
<title>XML Query</title>
<author>Fernandez</author>
<author>Suciu</author>
<summary>...</summary>
</book>
```

Different weights for query condition parts
Retrieval of XML trees

```xml
//*[ ... $c-word$ "Web"]
```

yields

```xml
<title rsv="0.8">Data on the Web</title>
<book year="1999" isbn="1-55860-622-X" rsv="0.6">
  <title>Data on the Web</title>
  <author>Abiteboul</author>
  <author>Buneman</author>
  <author>Suciu</author>
  <abstract> … </abstract>
</book>
<summary rsv="0.2"> … </summary>
```

- **Relevance ranking shall**
  - consider the structure of XML documents
  - deliver best matching documents as top of the ranking
Comparison of XML Query Languages

<table>
<thead>
<tr>
<th>General Requirements</th>
<th>XPath</th>
<th>XQuery</th>
<th>XIRQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema Awareness</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Flexible Types</td>
<td>-</td>
<td>+</td>
<td>?</td>
</tr>
<tr>
<td>Embedding</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Order Preservation</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Weighted Queries</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>XPath</th>
<th>XQuery</th>
<th>XIRQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Extraction and Reduction</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Combination and Restructuring</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>