Module 3

XML Processing

(XPath, XQuery, XUpdate)

Part 4: XQuery Update Facility + XQuery Scripting

Summary of lecture so far

- XML and XML Schema
  - serialization of data (documents + structured data)
  - mixing data from different sources (namespaces)
  - validity data (constraints on structure)
- XQuery
  - extracting, aggregating, processing (parts of) data
  - constructing new data; transformation of data
  - full-text search
- **Next: Updates and Scripting**
  - **bringing it all togheter!**

XQuery Update Overview

- Activity in W3C, now candidate recommendation
  - requirements, use cases, specification documents
- Use as transformation + DB operation (side-effect)
  - Preserve Ids of affected nodes! (No Node Construction!)
- Updates are expressions!
  - return "()" as result
  - in addition, return a *Pending Update List*
- Updates are fully composable with other expressions
  - however, there are semantic restrictions!
  - e.g., no update in condition of an if-then-else allowed
- Primitive Updates: insert, delete, replace, rename
- Extensions to other expr: FLWOR, TypeSwitch, ...
- Either updates or results, single snapshot per query

Examples

- delete nodes //book[@year lt 1968]
- insert node <author/> into //book[@ISBN eq "34556"]
- for $x in //book
  where $x/year lt 2000 and $x/price gt 100
  return replace value of node $x/price
  with $x/price-0.3*$x/price
- if ($book/price gt 200) then
  rename node $book as "expensive-book"

- Update expressions work on "node" or "nodes"
- Some implementations use older syntax – *do* operation

Language Extensions Overview

- New Expressions:
  - Insert: Insert new XML instances
  - Delete: Delete XML instances
  - Replace, Rename: Replace/Rename XML Instances
  - Transform: modify a copy an existing XDM
  - fn:put(): place an XDM instance into a file/location
- Changed (composition) expressions
  - FLWR: Bulk update
  - If: Conditional update
  - Typeswitch: Type-Based updates
  - Comma Expression: Updates Sequences
  - Function Defintion: Define updating functions

Composability

- *Insert, delete, rename*, *replace, and calls to updating functions* are expressions
- Classify expressions as
  - Simple: all XQuery 1.0 expressions
  - Updating: all new Update expressions
- Updating is *not* fully composable with the rest
  - Semantic, not syntactic restrictions
- Updating only allowed in control-flow expressions (see previous slide) + standalone
- Control-flow expression get class type from their "input", only same type allowed for all inputs (both branches of if updating or simple)

INSERT - Variant 1

- Insert a new element into a document

  insert node UpdateContent **into** TargetNode
- UpdateContent: any sequence of items (nodes, values)
- TargetNode: Exactly one document or element
  - otherwise ERROR
- Optionally, specify if to insert at the beginning or end
  - as last: Content becomes first child of Target
  - as first: Content becomes last child of Target
  - No position: no fixed position (honor other first/last inserts)
- Nodes in Content assume a new Id.
- Whitespace, Text conventions as in ElementConstruction of XQuery

INSERT Variant 1

- Insert new book in the library

```
insert  node <book> <title>Die wilde Wutz</title> </book>
into document("www.uni-bib.de")//bib
```

- Insert new book at the beginning of the library

```
insert  node <book> <title>Die wilde Wutz</title> </book>
as first into document("www.uni-bib.de")//bib
```

- Insert new attribute into an element

```
insert node (attribute age { 13 })
into document("persons.xml")//person[@name = "KD"]
```

INSERT - Variant 2

- Insert at a particular point in the document

  insert node UpdateContent **(after | before)** TargetNode
- UpdateContent: No attributes allowed!
- TargetNode: One Element, Comment or PI.
  - Otherwise ERROR
  - Must have parent
- Specify whether before or behind target
  - Before vs. After
- Nodes in Content assume new Identity
- Whitespace, Text conventions as ElementConstructors of XQuery

Insert - Variant 2

- Add an author to a book

insert node <author>Florescu</author>

before //article[title = "XL"]/author[. = "Grünhagen"]

DELETE

- Deletes nodes from a document

  delete (node | nodes) TargetNodes

- TargetNodes: Any sequence of nodes
- Delete XML papers.

  delete node //article[header/keyword = "XML"]

- (Snapshot semantics: compute Ids of nodes.)
- Deletes 2's from (1, 1, 2, 1, 2, 3) not possible
  - need to construct new sequence with FLWOR, sequence functions, ...

REPLACE

- Variant 1: Replace a node
  replace node TargetNode **with** UpdateContent
- Variant 2: Replace the content of a node
  replace value of node TargetNode **with** UpdateContent
- TargetNode: One node (with Id)
- UpdateContent: Any sequence of items
- Variant 2 keeps the node ID of TargetNode
- Whitespace and Text as with inserts.
- Many subtelties
  - in UpdateContent, replace document with its children
  - can only replace one node by another node (of similar kind)

RENAME

- Give a node a new name

  rename node Target as NewName

- Target must be attribute, element, or PI
- NewName must be an expression that evaluates to a QName (or castable)
- First author of a book is principle author:

  rename node //book[1]/author[1]
  as "principle-author"

TRANSFORM

- Update on streaming data

  copy Var := SExpr modify UExpr return RExpr

- Return all Java programmers, but without their salary

```
for $e in //employee[skill = "Java"] return
copy $je := $e
   modify delete node $je/salary
return $je
```

- SExpr: Source expression - what to update
- UExpr: Update expression - update
- RExpr: Return expression - result returned

Is this an updating expression?

Put

- Extension of the function library
- `fn:put($node as node(),`
  `$uri as xs:string) as empty-sequence()`
- Places `$node` onto the location identified by `$uri`
- `$node` has to be document or element
- External effects are implementation-defined

Conditional Update

- Adopted from XQuery if then else expression

```
if (condition) then
    SimpleUpdate
else
    SimpleUpdate
```

- No "mixing" possible: either both updating or neither
- Same for typeswitch()

Bulk Updates: FLWUpdate

- INSERT and REPLACE operate on ONE node!
- Idea: Adopt FLWR Syntax from Xquery
  (ForClause | LetClause)+ WhereClause? *SimpleUpdate*
  - *SimpleUpdate:* insert, delete, replace or empty
- Semantics: Carry out *SimpleUpdate* for every node bound by FLW.
  - Quiz: Does an OrderBy make sense here?

FLWUpdate - Examples

- "Müller" marries "Lüdenscheid".

```
    for $n in //article/author/lastname
    where  $n = "Müller"
replace  value of node $n with "Müller-Lüdenscheid"
```

- Value-added tax of 19 percent.

```
   for $n in //book
insert node attribute vat { $n/@price * 0.19 } into $n
```

Further Update Expressions

- Comma Expression
  - Compose several updates (sequence of updates)

```
for $x in //books
   return (delete node $x/price,
delete node $x/currency)
```

- Function Declaration + Function Call
  - Declare functions with PUL
  - Impacts optimization and exactly-once semantics

Pending Updates List + Update Conflicts

- Each updating expression produces PUL
- Contains list of update operations (target+data)
- Bulk+control flow expressions need to merge PULs and resolve conflicts:
  - two or more update of the same type on the same node: rename, replaceNode, replaceValue, replaceElementContent
  - Put on the same uri
  - Namespace definitions: insertAttributes, rename, replaceNodes

Snapshot Semantics

- Updates are applied at the very end
  - inserts are not visible during execution
  - avoids Halloween problem
  - allows optimizations (change order of updates)
- Three steps
  - evaluate expr; compose pending update list (PUL)
    - append "primitive" to PUL in every iteration of FOR
  - conformance test of PUL
    - avoid duplicate updates to same node (complicated rule)
    - avoids indeterminism due to optimizations
  - apply PUL (update primitives one at a time)

Halloween Problem

```
for $x in $db/*
return do insert $x into $db
```

- Obviously, not a problem with snapshot semantics.
- (SQL does the same!)

Implementations

- Supported by most XQuery implementations
  - XML DB
  - File system
- Many vendors have proprietary updates
  - developed before the working drafts were released
  - Sometimes older drafts ("do" syntax)
  - need time to adjust to W3C recommendation
  - need to guarantee compatibility for customers

XQuery Scripting

Observation

- Despite of XQuery and XQuery Updates, we still need Java, C#, PHP
  - implement user interfaces
  - write complex applications
- Once you start using Java, you are tempted to do everything in Java
- **Goal: Get rid of Java!!! All XQuery!**
  - XQuery Scripting: Extension of XQuery for script
- Several research projects (XQueryP, XL, XQuery!)
- W3C published first proposal in March 2008
- Major disagreements on technical/political areas
- Lecture presents 2008 W3C draft

XQuery Scripting Overview

- Relax restrictions on Updating Expressions
  - Updating Expressions are allow to return XDM
  - Allow updating expressions everywhere
- Sequential Expressions:
  - Fine-grained snapshots
  - Expressions with side effects (=> sequential)
  - Define order in which expressions are evaluated
- New "sequential" expressions
  - Apply
  - Assignment
  - Block
  - While
  - Exit

Overview on semantic changes

- Today update snapshot: entire query
- Change:
  - Apply updates on specific operations (vary snapshot scope)
- Sequential execution order:
  - FLWOR: FLWO clauses are evaluated first; result in a tuple stream; then Return clause is evaluated in order for each tuple. Side-effects made by one row are visible to the subsequent rows.
  - Function Call: Parameters before function
  - COMMA: subexpressions are evaluated in order
  - (UPDATING) FUNCTION CALL: arguments are evaluated first before body gets evaluated
- Semantics is deterministic because of the sequential evaluation order

Apply Expression

- Syntax:

  Expr ::= ApplyExpr | ConcatExpr

  ApplyExpr ::= (ConcatExpr ";")+

  ConcatExpr ::= ExprSingle ("," ExprSingle)*

- Semantics:
  - Apply updates at every “;”
  - Return XDM instance of last ApplyExpr, discard others

Assignment Expression

- Syntax:

  "set" $VarName ":=" ExprSingle

- Semantics:
  - Change the binding of the variable
  - Variable has to be external or declared in a block (no let, for, or typeswitch)
- Semantics deterministic because of the sequential evaluation order
  - ExprSingle has to be simple expression

Block expression

- Syntax:

```
        "{"  ( BlockDecl ";")* Expr "}"
BlockDecl :=
            ("declare" $VarName TypeDecl? (":=" ExprSingle) ?)?
            (","  $VarName TypeDecl? (":=" ExprSingle) ? )*
```

- Semantics:
  - Declare a set of updatable variables, whose scope is only the block expression (initialize in order)
  - Evaluate each expression (in order) and make the effects visible immediately
  - Allow all kind of expressions (only updating is forbidden in variable declarations)

Functions and blocks

- Blocks are the body of sequential functions
- We relax the fact the a function cannot update some nodes *and* return a value

```
declare sequential function local:prune($d as xs:integer) as
    xs:integer
{
    declare $count as xs:integer := 0;
    for $m in /mail/message[date lt $d]
    return {   delete node $m;
                set $count := $count + 1
            };
    exit returning $count
}
```

While expression

- Syntax:

  "while" "(" *exprSingle* ")" Block
- Semantics:
  - Evaluate the test condition
  - If "*true*" then evaluate the Block; repeat
  - If "*false*", continue after Block
  - Each iteration can cause side effects
  - Result from block must come via `exit returning`
- Very different loop to FLWOR
  - FLWOR has pre-defined number of bindings
  - While behaves more like a Java/C for/while
  - Could be expressed using recursive sequential functions

Control Flow: Exit

- Traditional semantics, nothing surprising
- *Exit returning*: early exit+return value from a block, function or query
- Hard(er) to implement in a "database" style evaluation engine
  - Because of the lazy evaluation

- *Break* (or *continue*) was dropped from proposal, as it caused too many problems (e.g. while can “broken”, for not)

Example

```
declare sequential function myNs:cumCost($projects) as element( )*
{
    declare $total-cost as xs:decimal :=0;
    for $p in $projects[year eq 2005]
    return
            {set $total-cost := $total-cost+$p/cost;
             <project>
                        <name>{$p/name}</name>
                        <cost>{$p/cost}</cost>
                        <cumCost>{$total-cost}</cumCost>
            <project>
            };
}
```

XQuery: self join or recursive function

XQuery Scripting usage scenarios (I)

- XQueryP programs in the browsers
  - We all love Ajax (the results). A pain to program. Really primitive as XML processing goes.
  - Embedding XQuery(P)/Scripting in browsers
  - XQuery(P)/Scripting code can take input data from WS, RSS streams, directly from databases
  - Automatically change the XHTML of the page

XQueryP usage scenarios (II)

- XQuery Scripting programs in the databases
  - Complex data manipulation executed directly inside the database
  - Takes advantage of the DB goodies, performance, scalability, security, etc
- XQuery Scripting programs in application servers
  - Orchestration of WS calls, together with data extraction for a variety of data sources (applications, databases, files), and XML data transformations
  - XML data mashups

Related work

- Programming for XML:
  - Extensions to other programming languages: Xlinq, ECMAScript, PhP, XJ, etc
  - Extensions to XQuery: XL, XQuery!, MarkLogic's extension
  - Re-purposing other technologies: BPEL
- Long history of adding control flow logic to query languages
  - 15 years of success of PL /SQL and others
  - SQL might have failed otherwise !

Summary

- Side-effects
  - change data without re-creating the data
  - data keeps its identity (stays the "same")
- Add scripting capabilities
  - assignment, error handling, visibility of updates
- How does that impact your project?
  - Do you still need Java/PHP? Hopefully, no
  - Prediction: 1 year, can do projects without Java
  - Prediction: 10 years, XQuery is the new Java
- Implementations: partial implementations of XQueryP/Scripting in MXQuery/Zorba