Module 10

XQuery Update, XQueryP

Disclaimer: Work in progress!!!
Summary of M1-M9

• 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

• Web Services and Mashups
  – remote procedure calls on the Web
    (message format, service interfaces, broker)

• Next: Updates and Scripting
  – bringing it all together!
XQuery Update Facility
XQuery Updates Overview

• Activity in W3C; work in progress (~two years)
  – 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 expr.
  – 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, ...
Examples

• do delete //book[@year lt 1968]

• do insert <author/> into //book[@ISBN eq “34556”]

• for $x in //book
  where $x/year lt 2000 and $x/price gt 100
  return do replace value of $x/price
    with $x/price-0.3*$x/price

• if ($book/price gt 200) then
  do rename $book as “expensive-book”

• The „do“ needed in syntax! (Don‘t ask, just do it!)
Overview

- Insert: Insert new XML instances
- Delete: Delete nodes
- Replace, Renam: Replace/Rename nodes
- FLWOR Update: bulk update
- Conditional Updates:
  - if - then - else
  - typeswitch
- Comma Expression
- Updating Functions
INSERT - Variant 1

• Insert a new element into a document
do insert InsertionSeq into TargetNode
• InsertionSeq: transform docs into their children
• TargetNode: Exactly one document or element
  – otherwise ERROR
• Specify whether to insert at the beginning or end
  – as last: InsertionSeq becomes first child of Target (default)
  – as first: InsertionSeq becomes last child of Target
• Nodes in InsertionSeq assume a new Id.
• Whitespace, Textconventions as in ElementConstruction of XQuery
INSERT Variant 1

• Insert new book at the end of the library
  ```
  into document(„www.uni-bib.ch“)//bib
  ```

• Insert new book at the beginning of the library
  ```
  as first into document(„www.uni-bib.ch“)//bib
  ```

• Insert new attribute into an element
  ```
  do insert (attribute age { 13 }, <parents xsi:nil = „true“/>)
  into document(„ewm.de“)//person[@name = „KD“]
  ```
INSERT - Variant 2

- Insert at a particular point in the document
  
  \textbf{do insert} InsertionSeq (\textbf{after} | \textbf{before}) TargetNode

- Subtleties in InsertionSeq
  - No attributes allowed after an element!
  - Document nodes are transformed into their children

- \textbf{TargetNode}: One Element, Comment or PI.
  - Otherwise ERROR

- Specify whether before or behind target
  - Before vs. After

- Nodes in InsertionSeq assume new Identity

- Whitespace, Text conventions as ElementConstructors of XQuery
Insert - Variant 2

• Add an author to a book

```
do insert <author>Florescu</author>
before //article[title = „XL“]/author[. eq „Grünhagen“]
```
INSERT - Open Questions

• Insert into schema-validated instances?
  – When and how to validate types?
  – What is the type of the updated instance?
• Insert (V2): TargetNode has no Parent?
  – Is that an error?
• TargetNode is empty?
  – Is that an error or a no-operation?
DELETE

• Delete nodes from an instance
  ```
  do delete TargetNodes
  ```
• TargetNodes: sequence of nodes (no values!)

• Delete XML papers.
  ```
  delete //article[header/keyword = "XML"]
  ```

• Deletes 2‘s from (1, 1, 2, 1, 2, 3) not possible
  – need to construct new sequence with FLWOR
REPLACE

• Variant 1: Replace a node
  
  do replace TargetNode with UpdateContent

• Variant 2: Replace the content of a node
  
  do replace value of TargetNode with UpdateContent

• TargetNode:  One node (with Id)

• UpdateContent:  Any sequence of items

• 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)
• Give a node a new name
  
  do rename 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:
  
  do rename //book[1]/author[1]
  as „principle-author“
Composability

• *Insert, delete, rename, replace, and calls to updating functions* are expressions
• They are *not* fully composable with the rest
  – Semantic, not syntactic restrictions
• Side-effecting expressions only allowed in
  – “return” clause of a FLWOR
  – “then” and “else” branches of a conditional
  – the body of a function
  – within a typeswitch or stand-alone
  – only in “control-flow” style expressions
Bulk Updates: FLWOR Update

• INSERT and REPLACE operate on ONE node!
• Idea: Adopt FLWOR Syntax from XQuery
  $(ForClause \mid LetClause)^+ \text{ WhereClause? OrderBy? return } SimpleUpdate$
  - $SimpleUpdate$: insert, delete, replace, or rename
• Semantics: Carry out $SimpleUpdate$ for every node bound by FLW.
  - Quiz: Does an OrderBy make sense here?
FLWOR Update - Examples

• „Müller“ marries „Lüdenscheid“.

  for $n in //article/author/lastname
  where $n/text() eq „Müller“ return do
  replace value of $n with „Müller-Lüdenscheid“

• Value-added tax of 19 percent.

  for $n in //book return do
  insert attribute vat { $n/@price * 0.19 } into $n
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 $\text{db}$/*
return do insert $x$ into $\text{db}$

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

• Adopted from XQuery‘s if then else expr.
  
  if (condition) then
    SimpleUpdate
  else
    SimpleUpdate
Transformations

- Update streaming data - create new instances

  \[\text{transform copy } \text{Var} := \text{SExpr\hspace{1em}}\text{modify } \text{UEExpr\hspace{1em}}\text{return } \text{REExpr}\]

- Delete salary of Java programmers

  \[\text{for } \text{e in } //\text{employee[skill = "Java"] return }\]
  \[\text{transform copy } \text{je} := \text{e\hspace{1em}}\text{modify do delete } \text{je/salary\hspace{1em}}\text{return } \text{je}\]

- **SExpr**: Source expression - what to transform
- **UEExpr**: Update expression - update
- **REExpr**: Return expression - result returned
Further Update Expressions

• Comma Expression
  – Compose several updates (sequence of updates)
    
    for $x$ in //books
    return do delete $x$/price, do delete $x$/currency

• Typeswitch Expression
  – Carry out updates depending on the type

• Function Declaration + Function Call
  – Declare functions with side-effects
  – Impacts optimization and exactly-once semantics
Implementations

• MXQuery (www.mxquery.org)
  – implements full XQuery Update Facility
  – but, limitations in how to bind data to update to variables
  – but, MXQuery only implements subset of XQuery
  – MXQuery is an α release; bleeding edge

• Most database vendors have a proprietary update language
  – developed before the working drafts were released
  – need time to adjust to W3C recommendation
  – need to guarantee compatibility for customers
XQueryP
Observation

• Despite of XQuery and XQuery Updates, we still need Java
  – implement user interfaces
  – call Web services; interact with other programs
  – expose functions as Web service
  – write complex applications

• Once you start using Java, you are tempted to do everything in Java (-> your projects :-) )

• Goal: Get rid of Java!!! All XQuery!
  – XQueryP: Extension of XQuery for scripting
XQueryP Overview

• Sequential Mode: Visibility of Updates
  – define order in which expressions are evaluated
  – fine-grained snapshot (update primitive)

• New expressions
  – Assignment, Block, While, Break, Continue, Return

• Error handling (try-catch)

• Graphs: references and de-referencing

• Web Service Import, Call, and Export
Sequential evaluation order

• Slight modification to existing rules:
  – 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.
  – COMMA: subexpressions are evaluated in order
  – (UPDATING) FUNCTION CALL: arguments are evaluated first before body gets evaluated

Required (only) if we add side-effects immediately visible to the program: e.g. variable assignments or single snapshot atomic updates; otherwise semantics not deterministic.
Reduce snapshot granularity

• Today update snapshot: entire query
• Change:
  – Every single atomic update expression (insert, delete, rename, replace) is executed and made effective immediately
  – The effects of side-effecting external functions are visible immediately
• Semantics is deterministic because of the sequential evaluation order (point1)
Sequential evaluation mode and the FLWOR

for $x$ in <expression/>
let $y := <expression/>
where <expression/>
order by <expression/>
return

<side-effecting expression/>

No side-effects are visible until here.
Adding new expressions

- Assignment expressions
- Block expressions
- While expressions
- Break, Continue, Return

- Only under sequential evaluation mode
Assignment Expression

• Syntax:
  “set” $VarName “:=“ ExprSingle

• Semantics:
  – Change the value of the variable
  – Variable has to be external or declared in a block
    (no let, for, or typeswitch)

• Updating expression

• Semantics is deterministic because of the sequential evaluation order
  – restricted side-effects in ExprSingle: only one side-effecting expression (primitive) allowed!
Block expression

• Syntax:

```
{“ ( BlockDecl “;”)* Expr (“;” Expr)* “}”
```

BlockDecl :=

```
(“declare” $VarName TypeDecl? (“:=“ ExprSingle) ?)?
(“,,” $VarName TypeDecl? (“:=“ ExprSingle) ? )*
```

• Semantics:
  – Declare a set of updatable variables, whose scope is only the block expression (in order)
  – Evaluate each expression (in order) and make the effects visible immediately
  – Return the value of the last expression

• Updating if body contains an updating expression

• Optional "atomic" makes updates in block all or nothing (nothing, if an error occurs)
Atomic Blocks

• Syntax:
  
  "atomic" "{ " . . . " }"

• Semantics:
  – If the evaluation of Expr does not raise errors, then result is returned
  – If the evaluation of Expr raises a dynamic error then no partial side-effects are performed (all are rolled back) and the result is the error

• Only the largest atomic scope is effective

• Note: XQuery! had a similar construct
  – Snap {…} vs. atomic {…}
Functions and blocks

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

```xml
declare updating 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 {
            do delete $m;
            set $count := $count + 1
        };
    $count
}
```
While expression

• Syntax:
  “while” “(" exprSingle ")" “return” expr

• Semantics:
  – Evaluate the test condition
  – If “true” then evaluate the return clause; repeat
  – If “false” return the concatenation of the values returned by all previous evaluations of return

• Syntactic sugar, mostly for convenience
  – Could be written using recursive functions
Break, Continue, Return

• Traditional semantics, nothing surprising
• *Break* (or *continue*) the closest FLWOR or WHILE iteration
• *Return*: early exit from a function body
• Hard(er) to implement in a “database” style evaluation engine
  – Because of the lazy evaluation
Example

declare updating 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
Putting everything together: the sequential mode

- New setter in the prolog
- Syntax:
  - "declare" "execution" "sequential"
- Granularity: query or module
- What does it mean:
  - Sequential evaluation mode for expressions
  - Single atomic update snapshot
  - Several new updating expressions (blocks, set, while, break, continue)
- If the query has no side-effects, sequential mode is irrelevant, and traditional optimizations are still applicable
Try-catch

• Errors in XQuery 1.0, Xpath 2.0, XSLT 2.0
  - `fn:error(err:USER0005, "Value out of range", $value)`

• Traditional design for try-catch
  try (target-expr)
  catch ( $name as QName1, $desc, $obj )
    return handler-expr1
  catch ( $name as QName2, $desc, $obj )
    return handler-expr2...
  default ( $name, $desc, $obj )
    return general-handler-expr

• Example
  let $x := expr
  return
    try ( <a>{ $x } </a> )
    catch (err:XQTY0024)
    return <a>...
Web Services

• WS are the standard way of sending and receiving XML data
• XQuery are the standard way to program the XML processing
• We should design them consistently, natural fit

<table>
<thead>
<tr>
<th>XQuery</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>service</td>
</tr>
<tr>
<td>functions/operations</td>
<td>operations</td>
</tr>
<tr>
<td>arguments</td>
<td>ports</td>
</tr>
<tr>
<td>values for arguments and</td>
<td>value for input and output</td>
</tr>
<tr>
<td>Result: XML</td>
<td>messages: XML</td>
</tr>
</tbody>
</table>

• XQueryP proposes:
  – A standard way of importing a Web Service into an XQuery program
  – A standard way of invoking a WS operation as a normal function
  – A standard way of exporting an XQuery module as a Web Service

• Many XQuery implementations already support this. We have to agree on a standard.
import service namespace ws=„urn:GoogleSearch“ from "http://api.google.com/GoogleSearch.wsdl";

declare execution sequential;

declare variable $result;
declare variable $query;

set $query := mxq:readLine();
set $result :=
ws:doGoogleSearch("olqddkdQFHIlwHMXPerc1KNm+FDcPUf", $query, 0,\10, fn:true(), "", fn:false(), ",", "UTF-8", "UTF-8");

<results query="${query}">
  {
    for $url in $result/resultElements/item/URL
    return data($url)
  }
</results>
Defining a Web Service

service namespace eth="www.ethz.ch" port:2001;
declare execution sequential;
declare function eth:mul($a,$b) {$a * $b};
declare function eth:add($a,$b) {$a + $b};
declare function eth:sub($a,$b) {$a - $b};
declare function eth:div($a,$b) {$a div $b};

• Calling that Web Service...

import service namespace ab="www.ethz.ch" from "http://localhost:2001/wsl";
ab:div(ab:sub(ab:mul(ab:add(1,2),ab:add(3,4)),1),5)
Bubblesort in XQueryP

declare execution sequential;
declare variable $data := (5,1,9,5,7,1,7,23,7,22,432,4,2,765,3);
declare variable $len := 15;
declare variable $changed := fn:true();
while($changed) return {
    declare $i := 1;
    set $changed := fn:false();
    while ($i < $len) return {
        if ($data[$i] > $data[$i + 1]) then {
            declare $cur := $data[$i];
            set $changed := fn:true();
            do replace $data[$i] with $data[$i+1];
            do replace $data[$i+1] with $cur }
        else();
        set $i := $i + 1 } };
Adding references to XML

- XML tree, not graph
- E/R model graph, not tree
- Inherent tension, XML Data Model is the source of the problem, not XQuery
- Example
  - let $x := <a><b/>a/> return <c>{$x/b}</c> /* copy of <b/>*/
- Nodes in XDM have node identifiers
  - Lifetime and scope of nodeids, implementation defined
- XQueryP solution:
  - fn:ref($x as node()) as xs:anyURI
  - fn:deref($x as xs:anyURI) as node()
- Lifetime and scope of URIs, implementation defined
- Untyped references (URIs)
- No changes required to:
  - XML Schema, XDM Data Model, Xquery type system
- **NOT YET IMPLEMENTED IN MXQuery!!!**
XQueryP usage scenarios

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

• XQueryP programs in the databases
  – Complex data manipulation executed directly inside the database
  – Takes advantage of the DB goodies, performance, scalability, security, etc

• XQueryP 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!

• This is certainly not new research, but a natural evolution

• Florescu, Kossmann: SIGMOD 2006 Tutorial
XQueryP Implementations

• Prototype in Big OracleDB
  – Presented at Plan-X 2005

• Prototype in BerkeleyDB-XML
  – Might be open sourced (if interest)

• MXQuery
  – http://www.mxquery.org  (Java)
  – Runs on mobile phones: Java CLDC1.1; some cuts even run CLDC 1.0
  – Eclipse Plugin available in March 2007

• Zorba C++ engine (FLWOR Foundation)
  – Small footprint, performance, extensibility, potentially embeddable in many contexts
XQueryP Pet Projects (at ETH)

• Airline Alliances
  – every student programs his/her own airline
  – form alliances
  – experiment: do this in Java/SQL first; then in XQueryP

• Public Transportation
  – mobile phone computes best route (S-Bahn)
  – integrate calendar, address book, ZVV, GPS

• Context-sensitive Remote Control
  – mote captures „clicks“ and movements
  – mobile phone determines context and action (TV, garage, ..)

• Lego Mindstorm
  – move to warmest place in a room

• Less of a toy (Oracle): XML Schema validator in XQueryP

• Your CS345b project goes here!
XQueryP Grammar (MXQuery)

- **Bold**: modifications to XQuery grammar rules
- **Italic**: new XQueryP grammar rules
LibraryModule::= (ModuleDecl | ServiceDecl) Prolog;
Setter ::= BoundarySpaceDecl | DefaultCollationDecl | BaseURIDecl | ConstructionDecl | OrderingModeDecl | EmptyOrderDecl | RevalidationDecl | CopyNamespacesDecl | ExecutionDecl;

ExecutionDecl ::= "declare" "execution" "sequential";
Import ::= SchemaImport | ModuleImport | ServiceImport;
QueryBody ::= SequentialExpr; (=> rewritten)
SequentialExpr ::= Expr(";" Expr)*;
PrimaryExpr ::= Literal | VarRef | ParenthesizedExpr | ContextItemExpr | FunctionCall | OrderedExpr | UnorderedExpr | Constructor | Block
FunctionDecl ::= "declare" "updating"? "function" QName "(" ParamList? ")" "as" SequenceType)? (Block | "external");
ExprSingle ::= FLWORExpr | QuantifiedExpr |
    TypeswitchExpr | IfExpr | InsertExpr | DeleteExpr | RenameExpr | ReplaceExpr | TransformExpr | AssignExpr |
    WhileExpr | TryExpr | OrExpr

Block ::= "atomic"? "{" (BlockDecl ";")* SequentialExpr
    ("return" ExprSingle| "continue"|"break")?"}";
BlockDecl ::= "declare" "$" VarName TypeDeclaration? ("="
    ExprSingle)? ("," "$" VarName TypeDeclaration? ("="
    ExprSingle)? )*;
AssignExpr ::= "set" "$" VarName "=" ExprSingle;
WhileExpr ::= "while" "(" ExprSingle ")" "return" ExprSingle;
TryExpr ::= "try" "(" ExprSingle ")" CatchExpr* (CatchExpr | DefaultCatchExpr);

CatchExpr ::= "catch" "(" ( "$" VarName ("as" NameTest)? ("," "$" VarName ("," "$" VarName)?)?)? ")" "return" ExprSingle;

DefaultCatchExpr ::= "default" "(" "$" VarName ("," "$" VarName ("," "$" VarName)?)?)? ")" "return" ExprSingle;

IfExpr ::= "if" "(" Expr ")" "then" (ExprSingle| "return" ExprSingle|"break"|"continue") "else ExprSingle|"return" ExprSingle|"break"|"continue";

TypeswitchExpr ::= "typeswitch" "(" Expr ")" CaseClause+ "default" ("$" VarName)? "return" (ExprSingle|"return" ExprSingle|"break"|"continue");

CaseClause ::= "case" ("$" VarName "as")? SequenceType "return" (ExprSingle|"return" ExprSingle|"break"|"continue");
• `ServiceImport ::= "import" "service" "namespace" NCName "=" URILiteral "from" URILiteral ("name" = NCName)?;`

• `ServiceDecl ::= "service" "namespace" NCName ="URILiteral" "port:" IntegerLiteral;`
Summary

- **Side-effects**
  - change data without re-creating the data
  - data keeps its identity (stays the „same“)
  - open questions concern „re-validation“ of data

- **Add scripting capabilities**
  - assignment, error handling, visibility of updates
  - Web Service calls; basic Mashups

- **How does that impact your project?**
  - Do you still need Java/PHP? Probably yes. :-(
  - Prediction: 1 year, can do projects without Java
  - Prediction: 10 years, XQuery(P) is the new Java

- **Implementations:** stay tuned :-(