

# XQuery 3.0

# Overview: XQuery 3.0

- Fix shortcomings of XQuery 1.0, not a radical change
- Better align XPath 3.0, XSLT 3.0, XQuery 3.0 (thus the version!)
- Properly incorporate some of the best ideas from other environments
  - Higher order functions (see Haskell, OCAML, ...)
  - Grouping, Outer Joins (SQL)
  - Windows (Stream Processing)
  - Error Handling (Programming Languages)
- Many small useful additions:
  - General FLWOR: Flexible composition
  - Switch Statement
  - Output Declarations
  - Formatting numbers and dates (XSLT)
  - Computed Namespaces
  - QNames with explicit URLs
  - Context Item Declaration, Default Values for External Variables
  - Function & Variable Annotations: private (OO programming), nondeterministic

# Higher Order Functions

- Common Feature of Functional Languages:  
function sort(data as item\*, comparator as function)  
...  
function compare-lexical (item, item)  
...  
sort (("a", "c", "b"), compare-lexical)
- Extend type system:
  - functions are now first-rate type, similar to nodes or atomic values
  - Type testing on Functions, type declarations on functions
- Constructor for function items:
  - Literal (to refer to existing)
  - Inline/Anonymous (to define ad-hoc)
- Dynamic Function Invocation

# HOF: Declaring

LiteralFunctionItem ::= QName "#" IntegerLiteral

Refer to a function with a name and # of parameters  
(Note: # parameters is sufficient, since XQuery functions are not polymorphic)

local:myfunc#2

fn:substring#3

InlineFunction ::= "function" "(" (ParamList? ")" ("as" SequenceType)? EnclosedExpr

Creates an anonymous function item

```
function($a as xs:double, $b as xs:double) as xs:double {  
  $a * $b }
```

# HOF: Invoking

PrimaryExpr "(" (ExprSingle ("," ExprSingle)\*)? ")" "

Invoke a function item produced by the primary expression, using the parameters given in the list

$\$f(2, 3)$  : call the function in  $\$f$  with the two parameters 2 and 3

$\$f[2]("Hi there")$  : call the second function in  $\$f$  with a single parameter "Hi there"

$\$f()[2]$  : call the function in  $\$f$  with no parameters, take the second value of the result

# HOF: Example

```
declare function local:sort( $seq as item()* , $key as
  function(item()) as xs:anyAtomicType ) as item()* {
  for $a in $seq
  order by $key($a)
  return $a };
```

```
local:sort(
  tokenize("The quick brown fox jumps over ...", " "),
  function($a) {lower-case($a)})
```

- More complex cases (maybe as exercises)
  - Nested sequences
  - Recursive transformations

# General FLWOR

- XQuery 1.0 FLWOR strict sequence of (for|let)\*, where, order by, return
- Not flexible enough for all the new extensions
- Relax to initial\_clause, (anything but return)\*, return
- Initial\_clause ::= for, let, for window
- Semantics: Each operation produces/consumes a stream of variable binding set (aka tuple), return maps back to XDM

```
for $x in ..., let $y in ..., let $z in ... =>  
($x = 1003, $y = "Fred", $z = <age>21</age>)  
($x = 1017, $y = "Mary", $z = <age>35</age>)  
($x = 1020, $y = "Bill", $z = <age>18</age>)
```

# Outer Joins

- SQL: Join, on elements without partner add a NULL value
- Example: Lecturer LEFT JOIN Lecture

Name	LecID
Kossmann	1
Tatbul	2
Fischer	3

LecID	Lecture
1	XML
3	NIS

Name	Lecture
Kossmann	XML
Tatbul	NULL
Fischer	NIS

- Cumbersome to write in XQuery 1.0, since for would not bind to empty sequences
- Introduced *allowing empty* into *for* clause:  
for \$lecture allowing empty in  
    \$lectures/lecture[@LecID = \$lecturer/@LecID]

# Group By

- Put items into logical units using value expression, perform operations on each unit separately  
SELECT storeno, sum(qty)  
FROM SALES  
GROUP BY storeno
- Used in nearly all SQL queries, albeit restricted to aggregations on groups (data model!)
- Introduce as part of FLWOR clause, fully composable with any FLWOR operation on group  
"group" "by" "\$"VarName ("," VarName)\*
- Partition/Rebind the variable previous binding:
  - Group Variables: Single Value, representing group key (or part of it)
  - Group Contents: Make a sequence for each variable, concatenating all individual values

# Group By: Semantics

```
($store = <storeno>S101</storeno>, $item = <itemno>P78395</itemno>)
```

```
($store = <storeno>S102</storeno>, $item = <itemno>P94738</itemno>)
```

```
($store = <storeno>S101</storeno>, $item = <itemno>P41653</itemno>)
```

```
($store = <storeno>S102</storeno>, $item = <itemno>P70421</itemno>)
```

## group by \$store

```
($store = <storeno>S101</storeno>,  
 $item = (<itemno>P78395</itemno>, <itemno>P41653</itemno>))
```

```
($store = <storeno>S102</storeno>,  
 $item = (<itemno>P94738</itemno>, <itemno>P70421</itemno>))
```

- Group Keys are computed by atomizing all grouping variables, must yield a single value each
- Group Keys are compared using eq, special care for () and NaN
- Order in each group
- Order of between Groups is implementations-dependent, use separate order by if necessary

# Group By: Examples

```
for $s in $sales
let $storeno := $s/storeno
group by $storeno
return
<store number="{ $storeno}" total-qty="{sum($s/qty)}"/>
```

Outcome: <store number="S101" total-qty="1550" />  
<store number="S102" total-qty="2125" />

```
let $x := 64000
for $c in //customer
let $d := $c/department where $c/salary > $x
group by $d
return
  <department name="{ $d}"> Number of employees earning more than ${ $x}
  is {count($c)} </department>
```

How does the result look if there is a sales department with three customers?

# Windows

- Create contiguous subsequences of XDM sequences:  
What was the average daily temperature of my office in the last 4 weeks?
- Orthogonal to grouping
  - Grouping split according to values
  - Window splits according to order
- Window Clause as part of FLWOR
- Full composability
  - No coupling to aggregates as in many streaming systems.
  - Nested windows possible
- Lays foundation to extend XQuery as an event/stream processing languages, can be complemented with an extension of XDM for infinite sequences
- (our claim to fame: proposed by ETH, published at VLDB 2007)

# Example: RSS Feed Filtering

## Blog postings

```
<item>...
  <author>Ghislain</author>...
</item><item>...
  <author>Peter</author>...
</item><item>...
  <author>Peter</author>...
</item><item>...
  <author>Peter</author>...
</item><item>...
  <author>Ghislain</author>...
</item>
```

## ➤ Not very elegant

- three-way self-join: bad performance + hard to maintain
- “Very annoying authors”: n postings = n-way join

## Return annoying authors: 3 consecutive postings

```
for $first at $i in $blog
let $second := $blog[i+1],
let $third := $blog[i+2]
where
  $first/author eq
  $second/author and
  $first/author eq
  $third/author
return $first/author
```

# New Window Clause: FORSEQ

- Extends FLWOR expression of XQuery
- Generalizes LET and FOR clauses

– LET  $\$x := \$seq$

- Binds  $\$x$  once to the whole  $\$seq$

– FOR  $\$x$  in  $\$seq$  ...

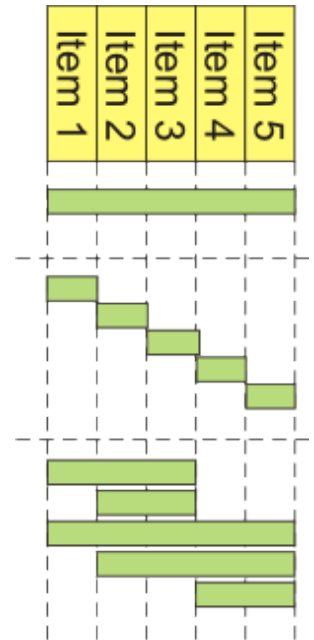
- Binds  $\$x$  iteratively to each item of  $\$seq$

– **FORSEQ  $\$x$  in  $\$seq$**

- Binds  $\$x$  iteratively to sub-sequences of  $\$seq$
- Several variants for different types of sub-sequences

- FOR, LET, FORSEQ can be nested

FLOWRExpr ::= (Forseq | For | Let)+ Where? OrderBy? RETURN Expr



# Four Variants of FORSEQ

WINDOW = contiguous sub-seq. of items

## 1. TUMBLING WINDOW

- An item is in zero or one windows (no overlap)

## 2. SLIDING WINDOW

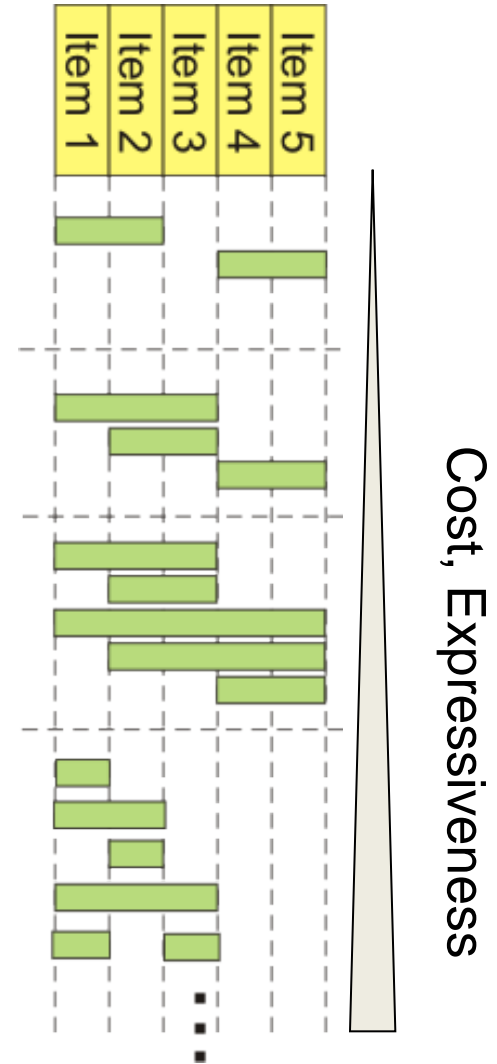
- An item is at most the *start* of a single window
- (but different windows may overlap)

## 3. LANDMARK WINDOW (not standard)

- Any window (contiguous sub-seq) allowed
- # windows quadratic with size of input

## 4. General FORSEQ (not standard)

- Any sub-seq allowed
- # sequences exponential with size of input!
- Not a window!



# RSS Example Revisited - Syntax

Annoying authors (3 consecutive postings) in RSS stream:

```
for tumbling window $window in $blog
  start curItem $first when fn:true()
  end nextItem $lookAhead when
  $first/author ne $lookAhead/author
where count($window) ge 3
return $first/author
```

- START, END specify window boundaries
- WHEN clauses can take any XQuery expression
- curItem, nextItem, ... clauses bind variables for whole FLOWR

# RSS Example Revisited - Semantics

```
<item><author>Ghislain</author></item>  
<item><author>Peter</author></item>  
<item><author>Peter</author></item>  
<item><author>Peter</author></item>  
<item><author>Ghislain</author></item>
```

```
For tumbling window $window  
start curItem $first when fn:true()  
end nextItem $lookahead when  
$first/author ne $lookahead/author  
where count($window) ge 3  
return $first/author
```

Open  
window

Closed  
+bound  
window

- Go through sequence item by item
- If window is not open, bind variables in start, check start
- If window open, bind end variables, check end
- If end true, close window, + window variables
- Conditions relaxed for sliding, landmark
- Simplified version; refinements for efficiency + corner cases

**=> Predicate-based windows, full generality**

# Error Handling

- XQuery 1.0 defines a broad set of errors (static and dynamic)
- Users can raise their own dynamic errors  
fn:error( \$code as xs:QName?, \$description as xs:string, \$error-object as item()\*)
- Yet, there is no way to deal with errors
  - Query terminates, caller needs to deal with it
  - Helper functions to work around: doc-available(), castable
- Mostly because the relation between error handling and updates was not clear, delayed to later revisions
- Complex applications need error handling!
- Introduce try/catch to deal with dynamic errors

# Try/Catch

```
TryCatchExpr ::= "try" "{" Expr "}" CatchClause+  
CatchClause ::= "catch" CatchErrorList CatchVars? "{" Expr "}"  
CatchErrorList ::= NameTest ("|" NameTest)*  
CatchVars ::= "(" CatchErrorCode ("," CatchErrorDesc (","  
CatchErrorVal)?)?)? ")"
```

- Capture error codes using QName and/or wildcards
- Multiple Catch expressions to handle different errors differently
- Optionally bind variables to the parts of error: code, description, object
- Applies only to lexical scope
- Rewrites still possible, but need to retain try/catch!

# Try/Catch Examples

- Capture everything, bind variables to error parts

```
try {  
  fn:error(fn:QName('http://www.w3.org/2005/xqt-errors',  
    'err:FOER0000'))  
}  
catch * ($errcode, $errdesc, $errval) { $errcode, $errdesc }
```

- Capture different error codes, handle in same way

```
try {  
  $x cast as xs:integer  
}  
catch err:FORG0001 | err:XPTY0004  
{ 0 }
```

# Switch Statement

- XQuery 1.0 has only typeswitch and if/then/else, but no value-based switch expression
- Syntax:
  - SwitchExpr ::= "switch" "(" Expr ")" SwitchCaseClause+  
"default" "return" ExprSingle
  - SwitchCaseClause ::= ("case" SwitchCaseOperand)+"return"  
ExprSingle
- Semantics:
  - Atomize switch and case expressions, must yield 0 or 1 value
  - Perform normal value comparison
  - First matching case expression is used
  - Errors only propagated for matching case

# Switch Examples

- `switch ($animal)`  
  `case "Cow" return "Moo"`  
  `case "Cat" return "Meow"`  
  `case "Duck"`  
  `case "Goose" return "Quack"`  
  `case lowercase($animal) return "Not audible"`  
  `default return "What's that odd noise?"`
- `switch(true())`  
  `case $a > 0 return "positive"`  
  `case $a < 0 return "negative"`  
  `default return "zero"`

# Output Declarations

- Separate Standards Document on how XQuery/XPath/XSLT serialize XDM into text, many options:
  - General Format: XML, XHTML, HTML, ...
  - Encoding
  - Intendation
  - ...
- No standard way how to access this functionality from XQuery (since serialization is optional)
- Use options in prolog and well-defined namespace:  
declare option output:method "xml";  
declare option output:encoding "iso-8859-1";  
declare option output:indent "yes";
- Only allowed in queries, not in modules/libraries

# Annotations

- We know *updating* from XQUF, *sequential, assignable* from Scripting
- XQuery 3.0 adds *(non)deterministic, private*
- And so on ...
- Generalize to arbitrary annotations
- "declare" [Annotation](#)\* ([VarDecl](#) | [FunctionDecl](#))
- Annotation ::= "%[EQName](#) ("[Literal](#) ("[Literal](#)"\*)")"?)
- Examples:
  - declare %private variable \$x
  - declare %java:method("java.lang.StrictMath.copySign") function smath:copySign(\$magnitude, \$sign) external;

# Namespaces

1) Recall the problem with context-sensitive names?

- Introduce QNames with explicit namespace URLs
- EQName ::= QName | URIQualifiedName
- URIQualifiedName ::= URILiteral ":" NCName
- e.g. <http://www.w3.org:localname>

2) How can dynamically create namespace bindings?

- Reintroducing namespace nodes!
- `<age xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"> { namespace xs {"http://www.w3.org/2001/XMLSchema"}, attribute xsi:type {"xs:integer"}, 23 }</age>`

# New Functions

- `serialize()/parse()`:  
Turn XDM into text and vice/versa
- `format-dateTime`, `format-date()`, `format-Time()`:  
detailed formatting with patterns, e.g. specific to country, calendar, language etc.
- `format-number()`, `format-integer()`:  
detailed formatting with patterns, e.g. custom decimal separator, optional digits ...
- `sin()`, `cos()`, `tan()`, `pi()`, `sqrt()`, ...:  
Basic trigonometry
- `partial-apply()`, `function-name()`, `function-arity()`:  
working on function items
- ...

# Status

- Currently working draft
- Feature set mostly stable, some ongoing smaller syntax changes (e.g. function annotations)
- Vendors beginning to pick up features, still at cherry-picking phase