ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Extending XQuery with Window Functions

Tim Kraska

ETH Zurich

Motivation

- XML is *the* data format for
  - communication data (RSS, Atom, Web Services)
  - meta data, logs (XMI, schemas, config files, ...)
  - documents (Office, XHTML, …)
- XQuery Data Model is a good match to streams
  - sequences of items
- XQuery has HUGE potential, **BUT ...**
  - poor *current* support for windowing (and continous queries)

Example: RSS Feed Filtering

Blog postings

```
<item>...
    <author>John</author>...
</item><item>...
    <author>Donald</author>...
</item><item>...
    <author>Donald</author>...
</item><item>...
    <author>Donald</author>...
</item><item>...
    <author>Peter</author>...
</item>
```

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
```

- Not very elegant
  - three-way self-join: bad performance + hard to maintain
  - “Very annoying authors“: n postings = n-way join

Other RSS Use-Cases

Blog postings

<item>...
<author>**John**</author>...
</item><item>...
<author>***Donald***</author>...
</item><item>...
<author>***Donald***</author>...
</item><item>...
<author>***Donald***</author>...
</item><item>...
<author>**Peter**</author>...
</item>

Queries

- Every hour provide summary of the RSS feed grouped by category, author etc.
- Every day provide list of interesting topics in the RSS feed. (e.g. title has to contain the word XQuery).
- ...

Toolbox – Data Smoothing

Data: Temperature data

<event time="1" temp="10"/>

<event time="2" temp="8"/>

<event time="3" temp="6"/>

<event time="3" temp="12" />

<event time="7" temp="10"/>

<event time="7" temp="10"/>

<event time="10" temp="10"/>

<event time="11" temp="20"/>

<event time="12" temp="14"/>

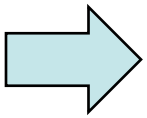
....

Queries

- Current Max/Min over time → 10, 10, 12, 12....
- Moving Average
  - time-based e.g. last 3 seconds → (9,10,12,...)
  - count-based: e.g. 4 last events → (9, 9, 9.5...)
- Exponential Smoothing
- Outlier detection (e.g. two times higher than previous second) → <event time="11"/>

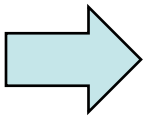
Positional Grouping

```
<q/>
<bullet>one</bullet>
<bullet>two</bullet>
<x/>
```



```
<q/>
<list>
    <bullet>one</bullet>
    <bullet>two</bullet>
</list>
<x/>
```

```
<nb>4</nb>
<nb>9</nb>
<nb>11</nb>
<nb>12</nb>
<nb>13</nb>
<nb>20</nb>
<nb>21</nb>
```



```
<nb>4</nb>
<nb>9</nb>
<nb>11-13</nb>
<nb>20-21</nb>
```

Application Areas

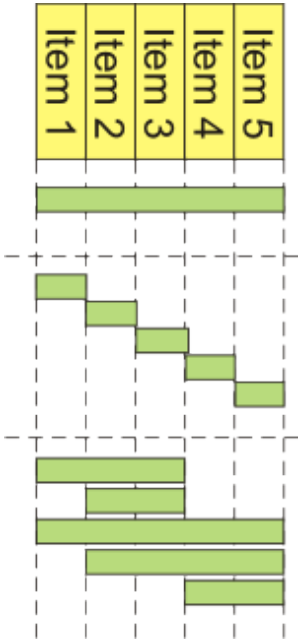
- Overall about 60 use cases specified
- Domains ranging over
  - RSS
  - Financial
  - Social networks/Sequence operations
  - Stream Toolbox
  - Document formatting/positional grouping

Overview

- Motivation and Use Cases
- **Windows for XQuery**
- Continuous XQuery
- Implementation and Optimization
- Linear Road Benchmark
- Summary + Future Work

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
  - **FOR WINDOW $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 ::= (FOR WINDOW| For | Let)+ Where? OrderBy? RETURN Expr

Two 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)

RSS Example Revisited

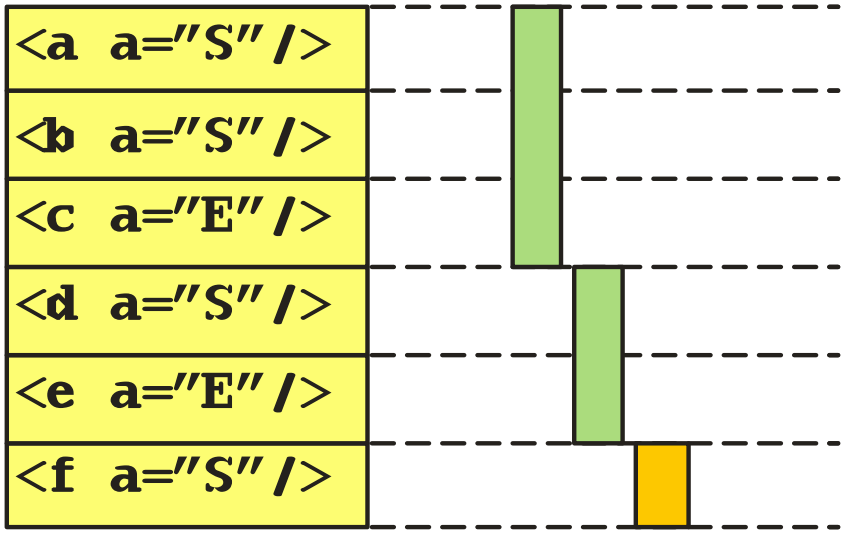
Annoying authors (3 consecutive postings) in RSS stream:

```
for tumbling window $window in $blog
    start $first when fn:true()
    end next $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
- START, END clauses bind variables for whole FLOWR

Tumbling Window - Syntax

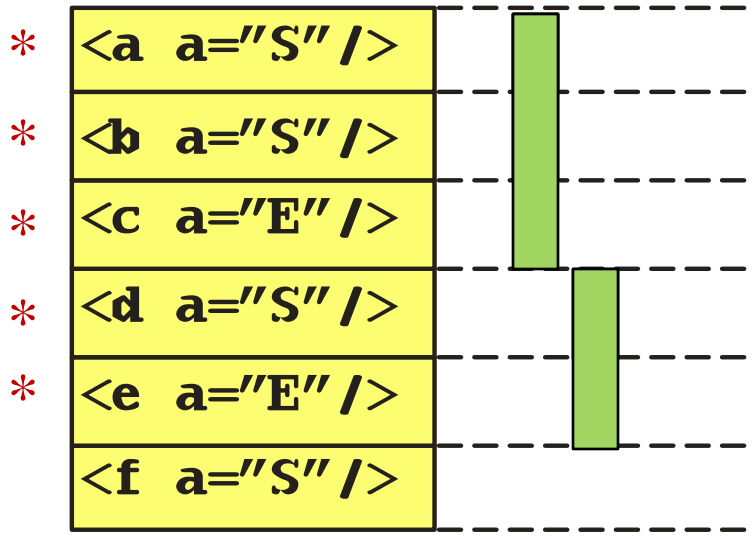
```
for tumbling window $w in $seq
  start at $x when $seq[$x]/@a eq "S"
  end at $y when $seq[$y]/@a eq "E"
return $w
```



Tumbling Window - Semantics

Evaluate to BEV

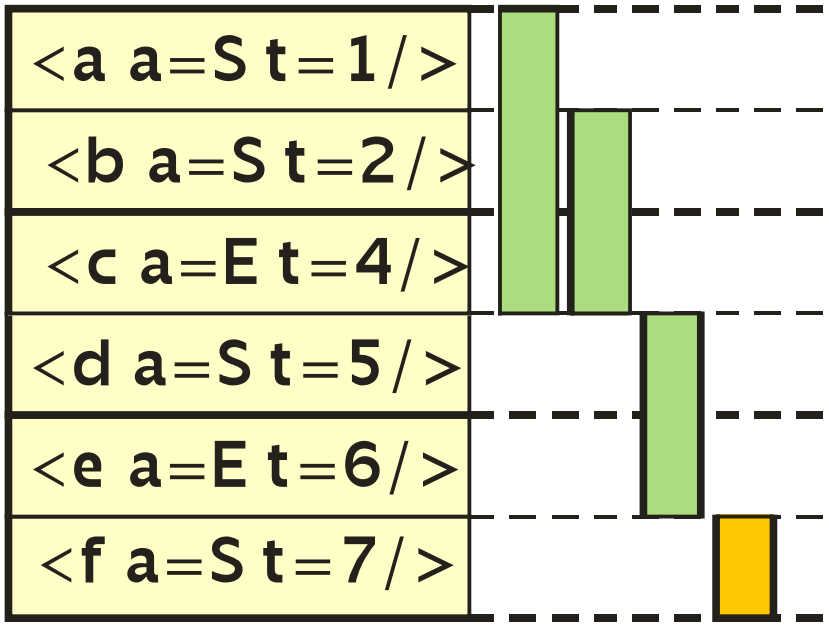
```
for tumbling window $w in $seq
  start at $x when $seq[$x]/@a eq "S"
  end at $y when $seq[$y]/@a eq "E"
return $w
```



- 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 the window and bind the variable

Sliding Window

```
for sliding window $w in $seq
   start at $x when $seq[$x]/@a eq "S"
   end at $y when $seq[$y]/@a eq "E"
return $w
```



Additional syntactic sugar

As the current, next and previous item are regularly needed, we introduced syntactic sugar for those

```
for tumbling window $w in $seq
start at $x
        when $seq[$x - 1] eq “A” and $seq[$x] eq “B”
end at $y when $seq[$y + 1] eq “C”
```

==

```
forseq $w in $seq tumbling window
start $cur at $x previous $prev
        when $prev eq “A” and $cur eq “B”
end at $y next $next when $next eq “B”
```

RSS Example Revisited

Annoying authors (3 consecutive postings) in RSS stream:

```
for tumbling window $w in $blog
    start $first when fn:true()
    end next $lookAhead when $first/author ne
            $lookAhead/author
where count($w) 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

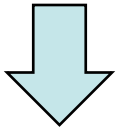
Tool-Box: Moving Average

```
for sliding window $w in $seq
    start $s when fn:true()
    end next $e when $e/tstamp - $s/tstamp gt ’PT1H’
return fn:avg($w/temp)
```

- Time-Based moving average:
  - Input: (infinite) time-ordered sequence of posting with rating
  - Output: (infinite) sequence of doubles (Average rating of the last hour)

Positional Grouping:

```
<q/>
<bullet>one</bullet>
<bullet>two</bullet>
<x/>
```

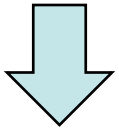


```
<q/>
<list>
    <bullet>one</bullet>
    <bullet>two</bullet>
</list>
<x/>
```

```
for tumbling window $w in $seq
    start $x when fn:true()
    end next $y when
        node-name($x) ne node-name($y)
return
    if ($x[self::bullet])
    then
        <list>
            {$w}
        </list>
    else
        $w
```

Positional Grouping: Page Ranges

```
<nb>4</nb>
<nb>9</nb>
<nb>11</nb>
<nb>12</nb>
<nb>13</nb>
<nb>20</nb>
<nb>21</nb>
```



```
<nb>4</nb>
<nb>9</nb>
<nb>11-13</nb>
<nb>20-21</nb>
```

```
for tumbling window $w in /doc/*
    start when fn:true()
    end $y next $z when ($y + 1) != $z
return
    if (count($w)=1) then
        $w
    else
        <nb>{data($x)} - {data($y)}</nb>
```

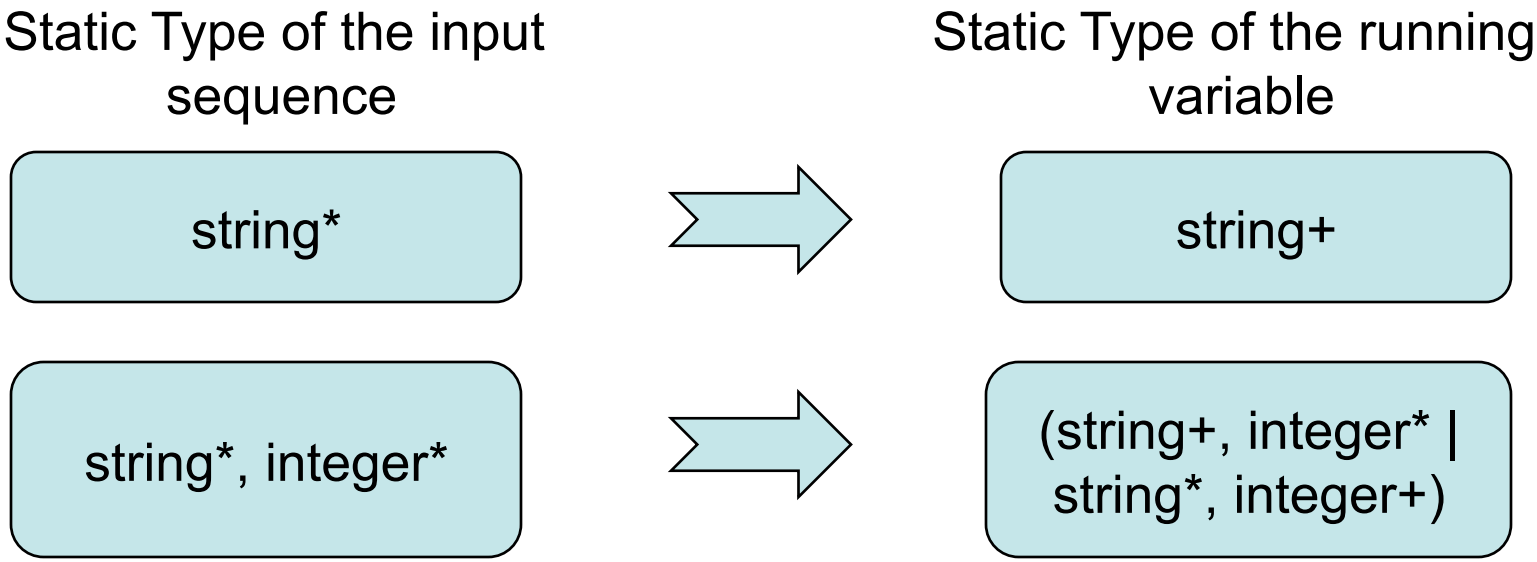
Fineprints

- What happens with last „window“ (END?)
  - Default: that window is nevertheless evaluated
    That is, EoS implicitly matches END condition
  - **for tumbling window start $s when ... end $e only when ...**
    - only use windows that fulfill END cond
- The windows are bound in ascending order of the position of the last item of a window
- Newstart for tumbling windows
  - There are several use cases in which the START condition should implicitly define the end of a window.
  - In order to implement such use cases, the WHEN condition of the END clause can be left out (means an implicit end)

Fineprints (ctd) - Typing

Typing

- Static typing for the Window clause is straightforward.
- Examples :



BNF

FLWORExpr	::=	(WindowClause|ForClause|LetClause) + WhereClause? OrderByClause? ”return” ExprSingle
WindowClause	::=	"for" TumblingWindowClause | SlidingWindowClause
TumblingWindowClause	::=	"tumbling" "window" "$" VarName TypeDeclaration? "in" ExprSingle StartCond EndCond?
SlidingWindowClause	::=	"sliding" "window" "$" VarName TypeDeclaration? "in" ExprSingle StartCond EndCond
StartCond	::=	"start" WindowVars "when" ExprSingle
EndCond	::=	"end" WindowVars "only"? "when" ExprSingle
WindowVars	::=	("$" VarName)? PositionalVar? ("previous" "$" VarName)? ("next" "$" VarName)?

Overview

- Motivation
- Windows for XQuery
- **Continuous XQuery**
- Implementation and Optimization
- Linear Road Benchmark
- Summary + Future Work

Continuous XQuery

- Streams are (possibly) infinite
  - e.g., a stream of sensor data, stock ticker, ...
  - not allowed in XQuery 1.0:
    infinite sequences are not part of XDM

=> **Proposed extension**

  - allow infinite sequences, new occurrence indicator: **
  - much less disruptive than SQL stream extensions

- Example: inform me when temperature > 0°C

```
declare variable $stream as (int)**;
for $temp in $stream
   where $temp ge 0 return <alarm/>
```

XQuery Semantics on Infinite Sequences

- Blocking expressions (e.g., ORDER BY)
  - not allowed, raise error
- Non-blocking expressions
  - infinite input -> infinite output (e.g., *If-then-else*)
  - infinite input -> finite output (e.g., *[5]*)
  - Some expressions undecidable at compile time (e.g., *Quantified expression*)

⇒We developed derivation rules for all expressions, similar to formalism of updating expressions

⇒Short version in the paper, extended version in a tech report (go to mxquery.org)

Overview

- Motivation
- Windows for XQuery
- Continuous XQuery
- **Implementation and Optimization**
- Linear Road Benchmark
- Summary + Future Work

Implementation Overview

- Window clause
  - parser: add new clause
  - compiler: some clever optimizations
  - runtime system: new iterators + indexing
- Continuous XQuery
  - parser: add ** occurrence indicator
  - context: annotate functions & operators
  - compiler: data flow analysis (infinite input)
  - optimizations at store, scheduler level possible!
- Easy to integrate
  - extended existing Java-based, open source engine

Optimization: Cheaper Window

Remember:

cost(tumbling) << cost(sliding)

```
for tumbling window $w in $seq
   start $s when $s eq „a"
   end $e when $e eq „c" ...
```

Assume (stream) schema knowledge:

a, b, c, a, b, c, ...

⇒ Only one open window possible at a time

⇒ Rewrite to tumbling

Additional Optimizations I

- Predicate Movearound
  - move predicates from *where* to *start*/*end*
  - reduce number of open/bound windows
  - need schema knowledge
- Indexing Windows
  - needed to handle large number of predicates and/or complex value predicates
  - speed up evaluation of END condition
  - index windows just like any other collection
  - keys on start variable values

Additional Optimizations II

- Improved Pipelining
  - start evaluating WHERE and RETURN clauses even though last item has not been read
- Hopeless Windows
  - detect windows that can never be closed
  - i.e., END condition is not satisfiable
- Aggressive Garbage Collection
  - Materialize only items needed for WHERE and RETURN clauses

Overview

- Motivation
- Windows for XQuery
- Continuous XQuery
- Implementation and Optimization
- **Linear Road Benchmark**
- Summary + Future Work

Linear Road Benchmark

- The only established streaming benchmark
- Models dynamic road pricing scenario
  - toll information, accidents, accounts, … as streams
  - historic queries on large database
- Complex workload
  - streams: window-based aggregation, correlation, ...
  - involves response time guarantees (< 5 sec)
  - load factor (L) determines number of inputs/second
- Compliant Implementations with Results
  - Aurora (L=2.5)
  - IBM Stream Processing Core (L=2.5 on single machine)
  - RDBMS (L=0.5): reference implementation by Aurora

Linear Road on MXQuery

- First attempt: One big XQuery expression
  - bad performance – we are not there, yet!
- Second attempt: 8 XQuery expressions
  - explicitly specify where to materialize
- Hardware (comparable to Aurora, IBM):
  - Linux box: 1 AMD Opteron 248 processor, 2.2 GHz
  - 2 GB main memory
  - Sun JVM Version 1.5.0.09
- Software: MXQuery engine (Java, open source)
  - stream data in main memory
  - historic data in MySQL database
  - no transactions, recoverability, security, etc.

Results

- L up to 2.0 fully compliant
  - we improved a bit since the paper was accepted
  - at L = 2.5: maximum response time 116 sec (5 sec allowed)
- Aurora, IBM compliant up to L= 2.5
- Why are we slower?
  - general-purpose XQuery engine vs. hand-written + hand-tuned query plan
  - No additional DSMS infrastructure (scheduler,…)
  - Java vs. C++ engine

⇒XQuery is not the problem!!!

Summary and Future Work

- XQuery 1.1 will have windowing
- Windowing covers also positional grouping
- XQuery is a good fit for CQ
- XQuery on streams is efficient
- Future Work
  - rigorous study of optimization techniques
  - stream schema

Backup

Partioning Windows?

- Aka „parallel tumbling“/ “splitting“/ predicate windows“
- Split stream into several streams using a predicate
- Proposed in relational streaming system
- Not orthogonal to GROUP BY
- Wait for Group By, see how or if FORSEQ + Group By can be combined to achieve same effect

FOR and LET with FORSEQ

- FORSEQ generalizes FOR and LET
- for $x in $seq ...

```
forseq $x in $seq tumbling window
   start when fn:true() end when fn:true()
...
```

- let $x := $seq ...

```
forseq $x in $seq tumbling window
   start when fn:true() end when fn:false()
...
```

- (Nevertheless: FOR and LET still needed.)

Memory Management