ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Extending XQuery with Window Functions

Irina Botan, Peter M. Fischer,
Dana Florescu*, Donald Kossmann,
Tim Kraska, Rokas Tamosevicius

ETH Zurich, Oracle*

# Elevator Pitch Version of this Talk

*XQuery* can do *stream processing* now, too!

- It is easy
  - Single new clause for window bindings
  - Simple extension of data model
- It is fast
  - Linear Road compliance L=2.0

# Motivation

- XML is *the* data format for
  - communication data (RSS, Atom, Web Services)
  - meta data, logs (XMI, schemas, config files, ...)
  - documents (Office, XHTML, …)
- XQuery is the way to process XML data
  - even if it is not perfect, it is has many nice abilities
  - works well for non-XML: CSV, binary XML, ...
- XQuery Data Model is a good match to streams
  - sequences of items
- XQuery has HUGE potential, **BUT ...**
  - poor *current* support for streams/continous queries

# Example: RSS Feed Filtering

## Blog postings

<item>...
  <author>**John**</author>...
</item><item>...
  <author>***Tom***</author>...
</item><item>...
  <author>***Tom***</author>...
</item><item>...
  <author>***Tom***</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

# Overcoming the Limitations of XQuery 1.0

- No (good) way to define a window
  - need to implement windows with self-joins
- No way to work on infinite sequences
  - infinite sequences are not in XQuery DM
  - no way to run continuous queries

=> **Goal of this work: Extend XQuery**

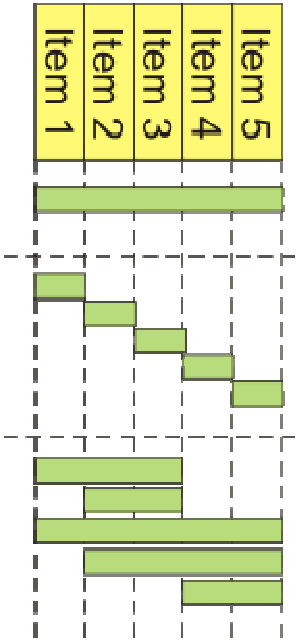
- new clause to express windows
- allow infinite sequences in XDM
- implement extensions in XQuery engine
- optimizations

# Overview

- Motivation
- **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
  - **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

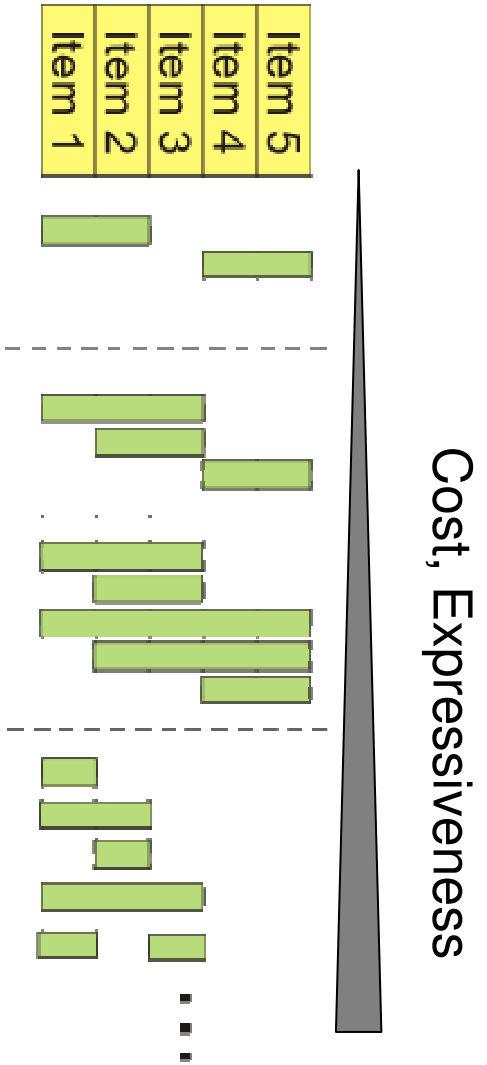
- An item is at most the *start* of a single window
- (but different windows may overlap)

## 3. LANDMARK WINDOW

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

## 4. General FORSEQ

- 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:

```
forseq $window in $blog tumbling window
    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

**Complete grammar in paper!**

# RSS Example Revisited - Semantics

<item><author>**John**</author></item>
<item><author>***Tom***</author></item>
<item><author>***Tom***</author></item>
<item><author>***Tom***</author></item>
<item><author>**Peter**</author></item>

```
forseq $window in $blog t          w
    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**

# Application Areas

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

=> Many use cases go beyond the abilities of relational streaming proposals

# 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

- FORSEQ 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) << cost(landmark)

```
forseq $w in $seq  sliding window
                   tumbling window
   start curItem $s when $s eq „a“
   end curItem $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!!!**

# Related Work

- Saxon's "item grouping" (XIME-P 06)
  - Designed for text document handling (XSLT UC)
  - No nested FORSEQ (multiple streams)
  - No sliding windows (only non-overlapping windows)
- SQL Extensions (on-going IBM/Oracle work)
- Semantics of Windows and Continuous Queries
  - STREAM, several surveys in DB + other communities
- Some recent research projects (SASE, Cayuga)
  - Invent language for smaller problem + fancy algorithm
  - By far not powerful enough

# Summary and Future Work

- Extending XQuery for streams/CQ is important
  - as important as the corresponding SQL extensions
- XQuery for programming streams is easy
  - need only small extensions (Windows, DM)
- XQuery on streams is efficient
  - no performance penalty for XQuery
  - XQuery as efficient as SQL!
- Future Work
  - rigorous study of optimization techniques
  - stream schema
  - more use cases and examples

# Thank you for your interest

# Questions?

Contact:
peter.fischer@inf.ethz.ch

Please visit http://mxquery.org for
- the MXQuery engine with FORSEQ
- the complete use case document
- tech report of infinite XDM semantics

# Backup Slides

# Time in XDM

- What about built-in time (system-time)?
  - XQuery DM is not temporal
  - Predicate-based approach allows time-based windows on application-defined timestamps
  - If needed, also possible to provide user-/system-defined function which generates timestamps for each incoming item

# Sliding Window: Moving Average

```
forseq $window in $seq sliding window
   start position $s when fn:true()
   end position $e when $e - $s eq 2
return fn:avg($w/rating)
```

- Input: (infinite) sequence of posting with rating
- Output: (infinite) sequence of doubles
  - Average rating of the last three postings
  - Alternative way to express Count-Based Window using position

# Time-Based Window: Web Log Analysis

```
declare variable $weblog as (entry)**
forseq $w in $weblog tumbling window
    start curItem $s when fn:true()
    end nextItem $e when
        $e/tstamp - $s/tstamp gt 'PT1H'
return fn:count($w[op eq "login"])
```

- Logins per hour
  - Express time constraints by (normal) predicates on time elements/attributes
  - As efficient as specific data model-based time
  - If „system“ time needed, inject it via function

# FORSEQ Post-Relational: Positional Grouping

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

=>

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

```
declare variable $seq;
forseq $w in $seq tumbling window
        start curItem $x when true()
        end nextItem $y when
                node-name($x) ne
                   node-name($y)
return
        if (string(node-name($x))
           eq "bullet") then
                <list>
                        {$w}
                </list>
        else
                $w
```

# General FORSEQ

- Compute all $2^n$-1 possible sub-sequences
  - Input: <a, b, c>
  - Bindings: { <a>, <a,b>, <b>, <a,b,c>, <a,c>, ... }
- Example: match regular expressions
  - Inform me if sensors „A (B|C) D“ have reported

```
forseq $w in $sensors
where $w[1]/id eq „A“ and
      ($w[2]/id eq „B“ OR $w[2]/id eq „C“) and
      $w[3]/id eq D
return ...
```

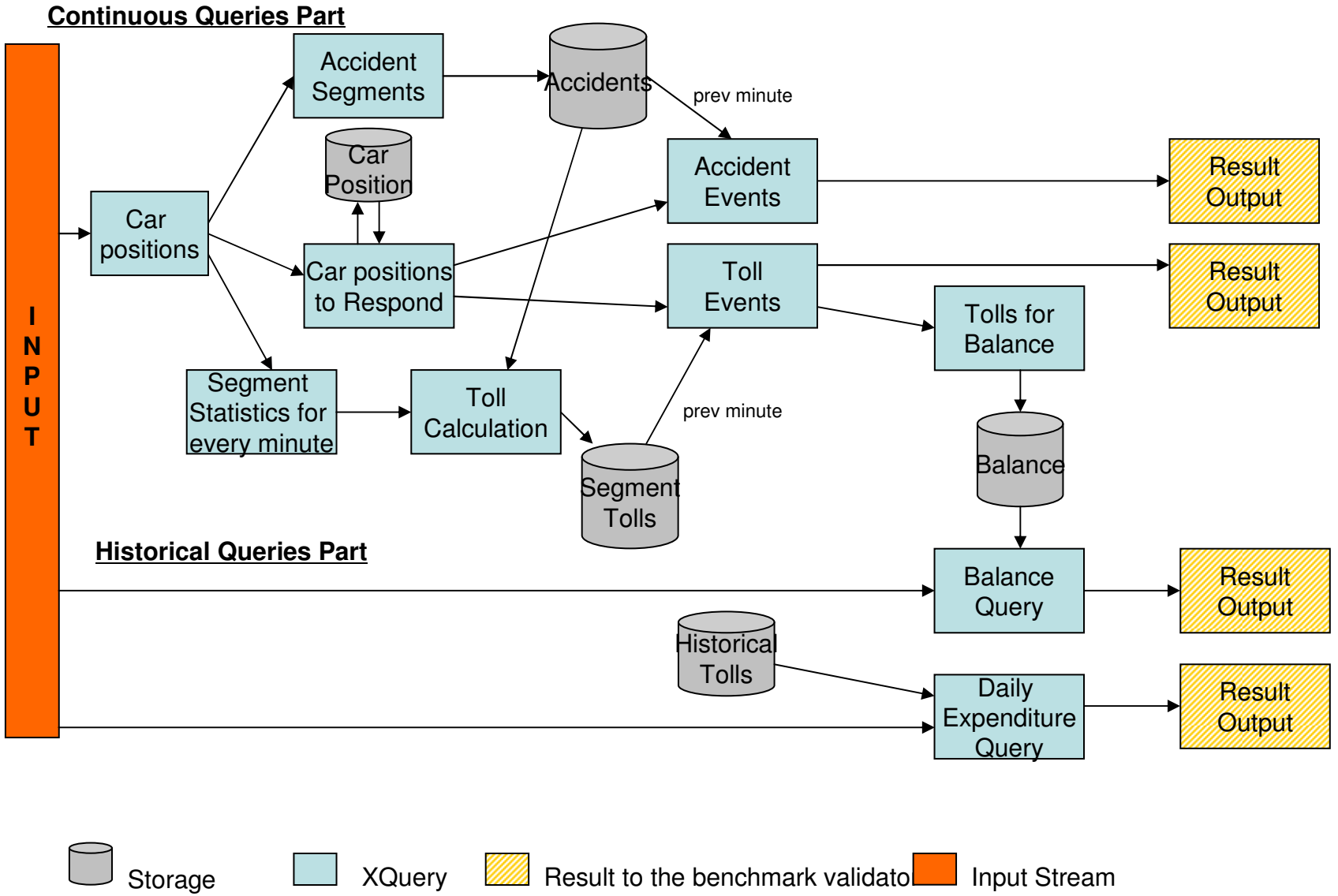
- (N.B.: XQuery Library has powerful reg. expr.)

# Why not use SQL?

- SQL does not work on XML data
  - even CSV might be a stretch
- SQL is too broken
  - extensions do not compose well (fresh start needed)
  - data model: relations to streams mapping is a mess

  -> careful (limited) SQL extensions for streams
- SQL works well for niche (maybe Wall Street), but not for masses (Web logs, text/media, ...)
- But, SQL extensions good foundation (thanks!)
  - use cases, window types, techniques very useful

# Linear Road Benchmark Implementation (data flow)



# Compatibility with XQuery + Extensions Proposals

- Groupby: orthogonal
  - *Forseq* partitions input into contiguous sequences using *position*
  - *Groupby* partitions input according to groups having certain *value*
  - Forseq *creates* bindings, GroupBy *uses* bindings
  - Maybe common proposal, but unusual behavior for both interest groups (SQL-GroupBy, Streaming)
  - But: We are open for alternative proposals

# 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

# MXQuery vs. RDBMS, (Stream) SQL vs. XQuery?

- Speed difference:
  - MXQuery Storage optimized for streaming data, not static data
  - Optimizations for streams available
- XQuery vs SQL:
  - Conceptually, neither is better in the areas which both support
  - Currently, implementations make the difference

# 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.)