

Windows for XQuery – Use Cases

Peter M. Fischer

Donald Kossmann

Tim Kraska

Rokas Tamosevicius

ETH Zurich, Switzerland
Email: {firstname.lastname}@inf.ethz.ch

Table of Contents

| | | |
|-----------|---|-----------|
| 1. | INTRODUCTION..... | 3 |
| 2. | SOCIAL GAMES: KEEPING TRACK OF PEOPLE..... | 5 |
| 2.1 | DATA..... | 6 |
| 2.2 | EVENTS IN A SEQUENCE..... | 8 |
| 2.3 | NEGATION & EVENTS IN A SEQUENCE..... | 11 |
| 2.4 | EVENTS WITHOUT SEQUENCE..... | 13 |
| 2.5 | AGGREGATES..... | 14 |
| 3. | DOCUMENT MANAGEMENT..... | 16 |
| 4. | FINANCIAL DATA..... | 21 |
| 4.1 | TRANSACTIONS..... | 21 |
| 4.1.1 | <i>List all open transactions.....</i> | <i>22</i> |
| 4.1.2 | <i>Most valuable Customers per day.....</i> | <i>24</i> |
| 4.1.3 | <i>Transaction time.....</i> | <i>25</i> |
| 4.1.4 | <i>Ship together.....</i> | <i>25</i> |
| 4.2 | STOCK DEALING (ALGORITHMIC TRADING)..... | 26 |
| 4.2.1 | <i>Stock Trend.....</i> | <i>26</i> |
| 4.2.2 | <i>Call-Put-Parity.....</i> | <i>27</i> |
| 5. | TOOLBOX..... | 29 |
| 5.1 | MOVING AVERAGE..... | 29 |
| 5.2 | EXPONENTIAL SMOOTHING..... | 30 |
| 5.3 | CURRENT MAX (MIN) VALUE..... | 34 |
| 5.4 | CURRENT SENSOR VALUE..... | 38 |
| 5.5 | STREAM SORTING..... | 40 |
| 5.6 | FILTER OUT REPETITIVE ALARMS..... | 41 |
| 5.7 | STREAM SYNCHRONIZATION..... | 42 |
| 5.8 | OUTLIER DETECTION..... | 46 |
| 6. | RSS..... | 48 |
| 6.1 | ANNOYING AUTHORS..... | 48 |
| 6.2 | SUMMARY BY CATEGORY..... | 49 |
| 6.3 | SUMMARY BY AUTHOR..... | 50 |
| 6.4 | INTERESTING TOPICS..... | 51 |
| 7. | LINEAR ROAD BENCHMARK..... | 52 |
| 8. | OTHER SCENARIOS..... | 59 |
| 9. | BIBLIOGRAPHY..... | 61 |

1. Introduction

This document describes the use cases which have driven the XQuery for Windows extension [1]. The use cases cover topics from different domains like RFID entry controls up to really complex financial cases such as detecting arbitrage possibilities in a financial stream. Although the sample inputs of these use cases are finite, they are all extensible to infinite data streams.

To show the compatibility with other extensions we have included additional examples which use GroupBy [2], XQuery Update [3] and XQueryP [4]. XQueryP is a small extension which enables XQuery expressions to exchange state information through variables. This extension makes it easier to develop applications in XQuery without relying on a host programming language.

As our proposed solution does not make use of implicit time, special attention has to be paid to the start of a stream. Assume the following sequence of elements:

```
<event time="2006-01-03T01:01:00-00:00">Event1</event>,
<event time="2006-01-03T01:10:00-00:00">Event2</event>,
<event time="2006-01-03T02:22:00-00:00">Event3</event>,
...
```

To create a 1 hour tumbling window over this sequence, one could use the following query:

```
declare variable $seq external;
forseq $w in $seq tumbling window
  start prevItem $p, curItem $c when
    hours-from-dateTime($p/@time) ne hours-from-dateTime($c/@time)
  end when newstart
return $w
```

Unfortunately, this query does not return the first hour. To overcome this, the query would need to be extended with an additional positional start.

```
declare variable $seq external;
forseq $w in $seq tumbling window
  start prevItem $p, curItem $c, position $p when
  hours-from-dateTime($p/@time) ne hours-from-dateTime($c/@time) or $p eq 1
  end when newstart
return $w
```

Alternatively, instead changing the original query additional dummy event could be added at the beginning of the sequence:

```
declare variable $originalSeq external;
declare variable $seq := (<event time="2000-01-01T00:01:00-00:00"/>,
  $originalSeq);

forseq $w in $seq tumbling window
  start prevItem $p, curItem $c when
  hours-from-dateTime($p/@time) ne hours-from-dateTime($c/@time)
```

```
    end when newstart  
return $w
```

To increase the readability of the use cases, we decided always apply the second option. Furthermore, we normally do not write the external variable declaration in the queries as we assume that the sample data is always bound to \$seq.

2. Social games: Keeping track of people

Most important for our extension are the entry gates use cases because they are very general and similar to a lot of other settings e.g. Web Log Auditing, RSS analysis or inventory control. The social games use cases assume a building with an entry control. Each person who gets into or out of the building has to use an identification card. A card reader corresponds to a sensor which continuously produces new events, which form the stream we want to query. For this scenario, several potential queries were created, which try to systematically cover different aspects of interest:

The first category “Events in a sequence” describes scenarios in which it is desired to know, that some events occur after others. Typical use cases include for example the information when B(arbara) comes later than A(nton) (test for concurrency of the sequence AB, in short SEQ(A B)), measuring the working time of Anton (SEQ(AIn, AOut)) or, more general, measuring the working time of every person in the building (SEQ(PIn ... POut) for every person P). The last case considerably increases complexity because persons do not enter or leave in the same order. Furthermore, queries with open starts and ends are part of this category, e.g. when Anton enters the building, give him a list of all persons who entered one hour before him (SEQ(1h before Ain, ?, Ain)).

The second category additionally takes into account negative events, i.e., events that do not happen before or after another event. Queries like “Inform boss if Anton does not come to work (between 5 am and 8 pm)” (SEQ(T5am, NOT(Ain), T8pm)) or “Inform me when Barbara enters the building if Anton is in the building” (SEQ(Ain, NOT(Aout), Bin)) are typical scenarios.

The third category leaves the area where events have to arrive in order, and tackles problems where an order is not necessarily required. Queries such as “Inform me when Anton and Barbara are in the office” (SEQ(Ain, NOT(Aout), Bin) or SEQ(Bin, NOT(Bout), Ain)) are typical examples.

The last category of entry gates use cases deals with aggregates over an infinite stream. Continuous queries like “Report each change of the number of people inside the building” as well as abnormal behavior detection like “Report if a person enters the building more than 5 times during 1 hour” can be found in this category.

Although the use cases might sound specific for the entry control, most of them have a general structure and are easily adoptable to other scenarios.

2.1 Data

The data consists of simple events containing a person name and a direction (in or out). As we don't use implicit time but only explicit time per event, you might want to get more precise or earlier results instead of waiting for the next arriving event. For these cases, we suggest mixing the incoming events with generated time events. The time events should be generated in the required granularity, like every minute, hour, day etc. For example, if there is only one employee, it might be difficult to formulate a query if this employee is not coming to work. Mixing generated time events with the incoming events, solves this problem. A sequence of events could look like

```
...
<event time="2006-01-03T01:00:00-00:00"/>,
<event time="2006-01-03T02:00:00-00:00"/>,
<event time="2006-01-03T03:00:00-00:00"/>,
<event time="2006-01-03T04:00:00-00:00"/>,
<event time="2006-01-01T04:30:00-00:00">,
  <person>Anton</person>
  <direction>in</direction>
</event>,
<event time="2006-01-03T05:00:00-00:00"/>,
...
```

Some use cases are easier to solve with this mixing, but we try to avoid relying on specific timestamps and assume that a lot of events are produced for a given day.

In the following, we present three different sets of data. These can also be viewed as one data stream only. However, as we need to compare the results, it is easier to do it for three different streams than for one large stream.

Day 1

```
<event time="2006-01-01T01:00:00-00:00"/>,
<event time="2006-01-01T10:30:00-00:00">
  <person>Anton</person>
  <direction>in</direction>
</event>,
<event time="2006-01-01T11:00:00-00:00">
  <person>Barbara</person>
  <direction>in</direction>
</event>,
<event time="2006-01-01T11:15:00-00:00">
  <person>Clara</person>
  <direction>in</direction>
</event>,
<event time="2006-01-01T12:15:00-00:00">
  <person>Clara</person>
  <direction>out</direction>
</event>,
<event time="2006-01-01T14:00:00-00:00">
  <person>Barbara</person>
  <direction>out</direction>
</event>,
<event time="2006-01-01T15:00:00-00:00">
  <person>Anton</person>
```

```
<direction>out</direction>
</event>,
<event time="2006-01-01T23:00:00-00:00"/>
```

Day 2

```
<event time="2006-01-02T01:00:00-00:00"/>
<event time="2006-01-02T10:30:00-00:00">
  <person>Barbara</person>
  <direction>in</direction>
</event>,
<event time="2006-01-02T11:00:00-00:00">
  <person>Anton</person>
  <direction>in</direction>
</event>,
<event time="2006-01-02T12:00:00-00:00">
  <person>Clara</person>
  <direction>in</direction>
</event>,
<event time="2006-01-02T14:00:00-00:00">
  <person>Barbara</person>
  <direction>out</direction>
</event>,
<event time="2006-01-02T23:00:00-00:00"/>
```

Day 3

```
<event time="2006-01-03T01:00:00-00:00"/>,
<event time="2006-01-03T09:00:00-00:00">
  <person>Doro</person>
  <direction>in</direction>
</event>,
<event time="2006-01-03T09:05:00-00:00">
  <gate>front door</gate>
  <person>Doro</person>
  <direction>out</direction>
</event>,
<event time="2006-01-03T09:10:00-00:00">
  <gate>front door</gate>
  <person>Doro</person>
  <direction>in</direction>
</event>,
<event time="2006-01-03T09:20:00-00:00">
  <gate>front door</gate>
  <person>Clara</person>
  <direction>in</direction>
</event>,
<event time="2006-01-03T09:25:00-00:00">
  <gate>front door</gate>
  <person>Doro</person>
  <direction>out</direction>
</event>,
<event time="2006-01-03T09:25:00-00:00">
  <gate>front door</gate>
  <person>Doro</person>
  <direction>in</direction>
</event>,
<event time="2006-01-03T09:30:00-00:00">
  <gate>front door</gate>
```

```

    <person>Doro</person>
    <direction>out</direction>
</event>,
<event time="2006-01-03T09:35:00-00:00">
    <gate>front door</gate>
    <person>Doro</person>
    <direction>in</direction>
</event>,
<event time="2006-01-03T17:00:00-00:00">
    <gate>front door</gate>
    <person>Clara</person>
    <direction>out</direction>
</event>,
<event time="2006-01-03T23:00:00-00:00"/>

```

2.2 Events in a Sequence

| |
|---|
| SEQ A, B |
| Notify me when Barbara enters the building later than Anton within 1 hour |
| Result for Day 1 |
| <alert>Barbara is later than Anton</alert> |
| Result for Day 2 |
| () |
| XQuery |
| <pre> forseq \$w in \$seq tumbling window start curItem \$x when \$x/person eq "Anton" and \$x/direction eq "in" end curItem \$y when \$y/@time - \$x@time gt xs:dayTimeDuration("PT1H") or (\$y/person eq "Barbara" and \$y/direction eq "in") where \$y/person eq "Barbara" return <alert>Barbara is later than Anton</alert> </pre> |
| EventsSequence/SEQ_AB_1hour_1.xq |
| <pre> forseq \$w in \$seq sliding window start curItem \$x when true end nextItem \$y, curItem \$z when \$y@time-\$x@time gt xs:dayTimeDuration("PT1H") let \$a := \$w[person eq "Anton" and direction eq "in"] let \$b := \$w[person eq "Barbara" and direction eq "in"] where \$x@time lt \$z@time return <alert>Barbara is later than Anton</alert> </pre> |
| EventsSequence/SEQ_AB_1hour_2.xq |
| Comments |
| <ul style="list-style-type: none"> - It doesn't matter if Anton is not anymore in the building - \$y and \$w[last()] are the same - Instead of \$a[1]/@time lt \$b[last()]/@time you could also use \$a[1] << \$b[last()] |

| |
|--|
| SEQ A, B |
| Measure how long Anton stayed in the building. |

| |
|---|
| Result for Day 1 |
| <time>PT4H30M</time> |
| Result for Day 2 |
| () |
| XQuery |
| <pre> forseq \$w in \$seq tumbling window start curItem \$x when \$x/direction eq "in" and \$x/person eq "Anton" force end curItem \$y when \$y/direction eq "out" and \$y/person eq "Anton" return <time>{\$y/@time - \$x/@time}</time> </pre> |
| EventsSequence/SEQ_AB_timein.xq |
| Comments |
| |

| |
|---|
| SEQ Sp...Ep with p=particular person |
| Measure the working time of each person |
| Result for Day 1 |
| <pre> <working-time> <person>Clara</person> <time>PT1H</time> </working-time>, <working-time> <person>Barbara</person> <time>PT3H</time> </working-time>, <working-time> <person>Anton</person> <time>PT4H30M</time> </working-time> </pre> |
| Result for Day 2 |
| <pre> <working-time> <person>Barbara</person> <time>PT3H30M</time> </working-time> </pre> |
| XQuery |
| <pre> forseq \$w in \$seq sliding window start curItem \$s and \$s/direction eq "in" end curItem \$e force when \$s/person eq \$e/person and \$e/direction eq "out" return <working-time> {\$seq[\$s]/person} <time>{\$e/@time - \$s/@time}</time> </working-time> </pre> |
| EventsSequence/SEQ_TimePerPerson.xq |
| Comments |
| |

SEQ (A, B, C) or SEQ (B, A, C)

| |
|--|
| Notify me when Clara enters the building later than Anton and Barbara within 1 hour |
| Result for Day 1 |
| <alert>Clara is later than Anton and Barbara</alert> |
| Result for Day 2 |
| () |
| Result for Day 3 |
| () |
| XQuery |
| <pre> forseq \$w in \$seq[direction eq "in"] where \$w[last()]/@time - \$w[1]/@time lt xs:dayTimeDuration("PT1H") and length(\$w) = 3 and ((\$w[1]/person eq "Anton" and \$w[2]/person eq "Barbara" and \$w[3]/person eq "Clara") or (\$w[2]/person eq "Anton" and \$w[1]/person eq "Barbara" and \$w[3]/person eq "Clara")) return <alert>Clara is later than Anton and Barbara</alert> </pre> |
| EventsSequence/SEQ_ABC_1.xq |
| <pre> forseq \$w in \$seq[direction eq "in"] sliding window start curItem \$x when \$x/person eq "Anton" or \$x/person eq "Barbara" end curItem \$y, nextItem \$z when \$z/@time - \$x/@time gt xs:dayTimeDuration("PT1H") or \$y/person eq "Clara" where \$y/person eq "Clara" and \$w/person eq "Anton" and \$w/person eq "Barbara" return <alert>Clara is later than Anton and Barbara</alert> </pre> |
| EventsSequence/SEQ_ABC_2.xq |
| <pre> forseq \$w in \$seq sliding window start curItem \$s when true end nextItem \$e, curItem \$z when \$e-\$s gt xs:dayTimeDuration("PT1H") let \$a := \$w[person eq "Anton" and direction eq "in"][1] let \$b := \$w[person eq "Barbara" and direction eq "in"][1] let \$c := \$w[person eq "Clara" and direction eq "in"][last()] where \$a/@time lt \$c/@time and \$b/@time lt \$c/@time and \$z/person eq "Clara" return <alert> Clara is later than Anton and Barbara </alert> </pre> |
| EventsSequence/SEQ_ABC_3.xq |
| Comments |
| <ul style="list-style-type: none"> - It doesn't matter wheter Anton or Barbara are not in the building anymore - It is not possible to use tumbling windows - Maybe the event stream has to be mixed with generated time events |

| |
|---|
| SEQ[???, A] – Q1 |
| Inform me about each person which comes 1 hour before Clara |
| Result for Day 1 |
| <person>Barbara</person> |
| Result for Day 2 |
| <person>Anton</person> |
| XQuery |
| <pre> forseq \$w in \$seq sliding window start curItem \$x when true </pre> |

| |
|---|
| <pre> end position \$y when \$seq[\$y+1]/@time -\$seq[\$x]/@time gt xs:dayTimeDuration("PT1H") or \$seq[\$y+1]/person eq "Clara" where \$seq[\$y+1]/person eq "Clara" return \$w[direction eq "in"]/person </pre> |
| EventsSequence/SEQ_xA.xq |
| Comments |
| If events can really arrive at the same time (including milliseconds) there might be a problem here |

| |
|--|
| SEQ[???, A] – Q2 |
| Inform me about each person which is at least 10 minutes earlier than Clara |
| Result for Day 1 |
| <person>Anton</person> |
| Result for Day 2 |
| <person>Barbara</person> |
| XQuery |
| <pre> forseq \$w in \$seq tumbling window start when true end curItem \$x when \$x/person eq "Clara" and \$x/direction eq in return distinct-values(for \$y in \$w where \$x/@time gt (\$y/@time + xs:dayTimeDuration("PT10M")) return \$x/person) </pre> |
| EventsSequence/SEQ_yA.xq |
| Comments |

| |
|--|
| (A... 1h later) |
| Inform me 1 hour after the “Anton” arrived |
| Result for Day 1 |
| <warning>Anton arrived 1h ago</warning> |
| Result for Day 2 |
| <warning>Anton arrived 1h ago</warning> |
| XQuery |
| <pre> forseq \$w in \$seq tumbling window start curItem \$x when \$x/person eq "Anton" force end nextItem \$y when \$y/@time - \$x/@time gt xs:dayTimeDuration("PT1H") return <warning>Anton arrived 1h ago</warning> </pre> |
| EventsSequence/SEQ_A1h.xq |
| Comments |

2.3 Negation & Events in a Sequence

| |
|--|
| SEQ(NOT(A)) |
| Inform boss if Anton does not come to work (between 5 am and 8 pm) |
| Result for Day 1 |

| |
|---|
| () |
| Result for Day 3 |
| <alert>Anton didn't come to work</alert> |
| XQuery |
| <pre> forseq \$w in \$seq[direction eq "in"] tumbling window start curItem \$s when hours-from-dateTime(\$s) ge 5 end nextItem \$e when hours-from-dateTime(\$e) ge 20 where not(\$w[person eq "Anton"]) return <alert>Anton didn't come to work</alert> </pre> |
| Negation/Seq_Not_A.xq |
| Comments |
| The number of events might be a problem. E.g. if Anton is the only employee nothing would be triggered, if he is not coming. → A solution is to introduce generated time events and mix them with the original event stream |

| |
|---|
| SEQ (NOT(A), B) |
| Inform me when Barbara enters the building and Anton is not in the building |
| Result for Day 1 |
| () |
| Result for Day 2 |
| <alert>Barbara: Anton is not in the building</alter> |
| XQuery |
| <pre> forseq \$w in \$seq/stream/event landmark window start position \$x when \$x eq 1 force end curItem \$y when \$y/person = "Barbara" and \$y/direction eq "in" let \$a_out := \$w[person eq "Anton" and direction eq "out"] let \$a_in := \$w[person eq "Anton" and direction eq "in"] where (empty(\$a_in)) or (\$a_out[last()] >> \$a_in[last()]) return <alert>Barbara: Anton is not in the building</alert> </pre> |
| Negation/Seq_Not_A_B_1.xq |
| <pre> forseq \$w in \$seq/stream/event tumbling window start position \$z, curItem \$x when (\$x/person eq "Anton" and \$x/direction eq "out") or \$z eq 1 end curItem \$y when \$y/direction eq "in" and \$y/person = ("Anton", "Barbara") where \$y/person eq "Barbara" return <alert>Barbara: Anton is not in the building</alert> </pre> |
| Negation/Seq_Not_A_B_2.xq |
| Comments |
| Might be a problem if we have no beginning event for Anton. But as Anton could also be already in the building this is fine. |

| |
|--|
| SEQ (A, NOT(B)) |
| Inform me when Anton enters the building and after that in 30 minutes Barbara does not. |
| Result for Day 1 |
| <alert>Barbara didn't come</alert> |
| Result for Day 2 |
| () |
| XQuery |
| <pre> forseq \$w in \$seq tumbling window start curItem \$x when \$x/person eq "Anton" and \$x/direction eq "in" end nextItem \$y when \$y/@time - \$x/@time > xs:dayTimeDuration("PT30M") where empty(\$w[person eq "Barbara"]) return <alert>Barbara didn't come</alert> </pre> |

| |
|---|
| Negation/Seq_A_Not_B.xq |
| Comments |
| <ul style="list-style-type: none"> - There are a lot of other ways to implement that - Also if Anton is already in the building nothing happens |

| |
|---|
| SEQ (Ain, NOT(Aout), Bin) |
| Inform me when Barbara enters the building if Anton is in the building. |
| Result for Day 1 |
| <alert>Barbara: Anton is inside the building</alert> |
| Result for Day 2 |
| () |
| XQuery |
| <pre> forseq \$w in \$seq tumbling window start curItem \$x when \$x/person eq "Anton" and \$x/direction eq "in" end curItem \$y when (\$y/person eq "Anton" and \$x/direction eq "out") or (\$y/person eq "Barbara" and \$x/direction eq "in") where \$y/person eq "Barbara" return <alert>Barbara: Anton is inside the building </alert> </pre> |
| Negation/Seq_A_Not_A_B.xq |
| Comments |
| |

2.4 Events without Sequence

| |
|---|
| A and B in |
| Notify me when Anton and Barbara entered office together (within 30 minutes). |
| Result for Day 1 |
| <alert>Anton and Barbara just arrived </alert> |
| Result for Day 3 |
| () |
| XQuery |
| <pre> forseq \$w in \$seq[direction eq "in"] tumbling window start curItem \$x when \$x/person = ("Barbara", "Anton") end nextItem \$y when \$y@time-\$x@time gt xs:dayTimeDuration("PT30M") where \$w/person eq "Anton" and \$w/person eq "Barbara" return <alert>Anton and Barbara just arrived </alert> </pre> |
| NoSeq/NOSEQ_BothIn.xq |
| Comments |
| - We don't use a smaller interval because of our provided data |

| |
|---|
| (SEQ Ain, NOT(Aout), Cin) or (SEQ Cin, NOT(Cout), Ain) |
| Notify me whether both Anton and Clara are in the office. |
| Result for Day 1 |
| <alert>Anton and Clara are in the office</alert> |
| Result for Day 3 |
| () |
| XQuery |
| let \$persons := ("Anton", "Clara") |

| |
|--|
| <pre> forseq \$w in \$seq/stream/event tumbling window start curItem \$x when \$x/person = \$persons and \$x/direction eq "in" end curItem \$y when (\$y/person = \$persons and \$x/person ne \$y/person and \$y/direction eq "in") or ((\$x/person eq \$y/person) and \$y/direction eq "out") where not(\$y[direction eq "out"]) return <alert>Anton and Clara are in the office</alert> </pre> |
| <pre> NoSeq/NOSEQ_BothThere_1.xq let \$persons := ("Anton", "Clara") forseq \$w in \$seq sliding window start curItem \$y when \$y/person = \$person end when false where count(\$w[person eq "Anton"][direction eq "in"]) eq 1 and count(\$w[person eq "Clara"][direction eq "in"]) eq 1 and count(\$w[person eq "Anton"][direction eq "out"]) eq 0 and count(\$w[person eq "Clara"][direction eq "out"]) eq 0 return <alert>Anton and Clara are in the office</alert> </pre> |
| <pre> NoSeq/NOSEQ_BothThere_2.xq let \$persons := ("Clara", "Anton") forseq \$w in \$seq/stream/event sliding window start curItem \$x when \$x/person = \$persons and \$x/direction eq "in" end curItem \$y when \$y/person = \$persons and \$y/person ne \$x/person where \$w[person eq "Anton"] [last()]/direction eq "in" and \$w[person eq "Clara"] [last()]/direction eq "in" return <alert>Anton and Clara are in the office</alert> </pre> |
| <pre> NoSeq/NOSEQ_BothThere_3.xq </pre> |
| <p>Comments</p> |

2.5 Aggregates

| |
|---|
| <p>3 times entering</p> |
| <p>Inform me when a person enters the building at least 3 times within 1 hour</p> |
| <p>Result for Day 1</p> |
| <p>()</p> |
| <p>Result for Day 3 (Attention: Not complete correct)</p> |
| <pre> <alert>Doro is suspicious<alert> </pre> |
| <p>XQuery</p> |
| <pre> forseq \$w in \$seq sliding window start curItem \$s when true end nexItem \$e when \$e@time-\$s@time gt xs:dayTimeDuration("PT1H") where count(\$w[person eq \$s/person]) ge 3 return <alert>{\$s/person} is suspicious<alert> </pre> |
| <pre> Aggregates/AGG_Enter_3X_1.xq </pre> |
| <pre> forseq \$w in \$seq landmark window start curItem \$s when true force end curItem \$e when \$e@time-\$s@time le xs:dayTimeDuration("PT1H") where count(\$w[person eq \$s/person]) eq 3 return <alert>{\$s/person} is suspicious<alert> </pre> |
| <pre> Aggregates/AGG_Enter_3X_2.xq </pre> |
| <p>Evaluation</p> |

| |
|--|
| Both queries could produce several repetitive results; Therefore a further filtering with “distinct values” would be helpful |
| Comments |
| |

| |
|--|
| 5 times entering per hour |
| For each hour present the list of people, who entered the building at least 5 times during that hour (each event with entering the building data should be evaluated once). |
| Result for Day 1 |
| () |
| Result for Day 3 |
| <alert>Doro is suspicious</alert> |
| XQuery |
| <pre> forseq \$w in \$seq tumbling window start curItem \$s when true end nextItem \$e when hours-from-dateTime(\$e@time) ne hours-from-dateTime(\$s@time) return <result> <time> from {\$s} to {\$e}</time> { for \$p in distinct-values (\$w/person/text()) where count(\$w[person eq \$p]/person) gt 5 return <alert> Person {\$p} is suspicious </alert> } </result> </pre> |
| Aggregates/AGG_Enter_List.xq |
| Comments |
| |

| |
|---|
| 3 people inside the building |
| Notify me when at least 3 people are inside the building |
| Result for Day 1 |
| <alert time="2006-01-02T12:00:00-00:00">3 different people in the building</alert> |
| Result for Day 3 |
| () |
| XQuery |
| <pre> forseq \$w in \$seq landmark window start position \$x when \$x eq 1 end curItem \$y when true where count(\$x[direction eq "in"]) gt count(\$x[person eq "out"]) + 2 return <alert time={\$y@time}>3 different people in the building</alert> </pre> |
| Aggregates/AGG_Inside_Count.xq |
| Comments |
| The “in” and “out” equation doesn’t hold if the stream is already running. In this case there might be for example more “out” than “in” events |

| |
|---|
| Notifying about people inside the building every hour |
| Notify me every hour about the number of people inside the building |

| Result for Day 1 |
|---|
| <pre><alert>0 people are in the building</alert> <alert>1 people are in the building</alert> <alert>3 people are in the building</alert> <alert>2 people are in the building</alert> <alert>1 people are in the building</alert> <alert>0 people are in the building</alert></pre> |
| Result for Day 3 |
| <pre><alert>0 people are in the building</alert> <alert>1 people are in the building</alert> <alert>2 people are in the building</alert> <alert>3 people are in the building</alert> <alert>2 people are in the building</alert> <alert>2 people are in the building</alert></pre> |
| XQuery |
| <pre>forseq \$w in \$seq landmark window start position \$x when \$x eq 1 end curItem \$y, nextItem \$z when hours-from-dateTime(\$y@time) ne hours-from-dateTime(\$z@time) let \$p:={ count(\$w[direction eq "in"]) - count(\$w[person eq "out"])} return <alert>{p}people are in the building</alert></pre> <p>Aggregates/AGG_Inside_Count_Rec.xq</p> |
| Comments |
| <p>Same “in”/”out” equation problem like in the use case before.</p> |

3. Document Management

These use cases are from Michael Kay’s paper “Positional Grouping in XQuery” [5] to show, that our language extension can also be used to solve those problems.

| Headings and Paragraphs |
|--|
| Convert a structure with implicit section to a structure with explicit sections |
| Input |
| <pre><body> <h2>heading1</h2> <p>para1</p> <p>para2</p> <h2>heading2</h2> <p>para3</p> <p>para4</p> <p>para5</p> </body></pre> |
| Output |
| <pre><chapter> <section title="heading1"> <para>para1</para> <para>para2</para> </section> <section title="heading2"> <para>para3</para> <para>para4</para> <para>para5</para></pre> |

| |
|--|
| <pre> </section> </chapter> </pre> |
| <p>XQuery</p> <pre> declare variable \$seq external; <chapter> { forseq \$w in \$seq/body/* start curItem \$s when string(node-name(\$s)) eq "h2" end when newstart return <section title="{data(\$s)}"> {for \$y at \$p in \$w where \$p > 1 return <para>{data(\$y)}</para> } </section> } </chapter> </pre> <p>PositionalGrouping/head_para.xq</p> |
| <p>Comments</p> |

| |
|--|
| <p>Adjacent Bullets</p> <p>The problem here is to identify a sequence of adjacent <bullet> elements (among a sequence containing any other kind of element) and wrap them in a containing <list> element.</p> |
| <p>Input</p> <pre> <p/> <q/> <bullet>one</bullet> <bullet>two</bullet> <x/> <y/> </pre> |
| <p>Output</p> <pre> <p/> <q/> <list> <bullet>one</bullet> <bullet>two</bullet> </list> <x/> <y/> </pre> |
| <p>XQuery</p> <pre> declare variable \$seq external; <doc>{ forseq \$w in \$seq/doc/* 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 }</doc> </pre> |

PositionalGrouping/adj_bullets.xq

Comments

Term Definition Lists

Within a glossary in HTML, a defined term (<dt>) can be followed by a definition <dd>. The task is to group these together within a <term> element. To make things more complicated, a group can consist of one or more <dt> elements followed by one or more <dd> elements.

Input

```
<dt>XML</dt>
<dd>Extensible Markup Language</dd>
<dt>XSLT</dt>
<dt>XSL Transformations</dt>
<dd>A language for transforming XML</dd>
<dd>A specification produced by W3C</dd>
```

Output

```
<term>
  <dt>XML</dt>
  <dd>Extensible Markup Language</dd>
</term>
<term>
  <dt>XSLT</dt>
  <dt>XSL Transformations</dt>
  <dd>A language for transforming XML</dd>
  <dd>A specification produced by W3C</dd>
</term>
```

XQuery

```
declare variable $seq external;
<doc>{
  forseq $w in $seq/doc/* tumbling window
    start curItem $x when string(node-name($x)) eq "dt"
    end curItem $y, nextItem $z when string(node-name($y)) eq "dd" and
      string(node-name($z)) eq "dt"
  return
    <term>
      {$w}
    </term>
}</doc>
```

PositionalGrouping/term_def_list.xq

Comments

- A really nice use case for tumbling windows
- Also this use case doesn't work correctly with force because then the last window does not bind to \$w. The end-of-stream binds the last window although the end expression doesn't match.

Continuation Markers

Concatenate a sequence of fragments marked with the attribute cont="yes" to indicate that the next fragment is a continuation.

Input

```
<in cont="yes">One way to</in <in cont="yes"> understand positional grouping is
<in> as an exercise in parsing.</in>
<in cont="yes">To get from a sequence of items</in>
```

| |
|--|
| <pre><in cont="yes"> to a tree, we could use</in> <in> some kind of grammar.</in></pre> |
| Output |
| <pre><para>One way to understand positional grouping is as an exercise in parsing.</para> <para>To get from a sequence of items to a tree, we could use some kind of grammar.</para></pre> |
| XQuery |
| <pre>forseq \$w in \$seq tumbling window start when true end curItem \$x when empty(\$x[@cont eq "yes"]) return <para>{data(\$w)}</para></pre> <p>PositionalGrouping/cont_markers.xq</p> |
| Comments |
| |

| |
|--|
| Page Ranges |
| Given a sequence of page references, identify sub-sequences that denote continuous ranges of page numbers. |
| Input |
| <pre><nb>4</nb> <nb>6</nb> <nb>9</nb> <nb>11</nb> <nb>12</nb> <nb>13</nb> <nb>18</nb> <nb>20</nb> <nb>21</nb></pre> |
| Output |
| <pre><nb>4</nb> <nb>6</nb> <nb>9</nb> <nb>11-13</nb> <nb>18</nb> <nb>20-21</nb></pre> |
| XQuery |
| <pre>declare variable \$seq external; <doc>{ forseq \$w in \$seq/doc/* tumbling window start curItem \$x when true() end curItem \$y, nextItem \$z when (\$y + 1) != \$z return if (count(\$w)=1) then \$w else <nb>{data(\$x)} - {data(\$y)}</nb> } }</doc></pre> <p>PositionalGrouping/page_range.xq</p> |
| Comments |
| |

| Arrange Rows |
|---|
| Arrange a sequence of items in fixed size columns of a table. (The same problem occurs when grouping records say ten to a page). |
| Input |
| <pre><data>Green</data> <data>Pink</data> <data>Lilac</data> <data>Turquoise</data> <data>Peach</data> <data>Opal</data> <data>Champagne</data></pre> |
| Output |
| <pre><table> <tr> <td>Green</td><td>Pink</td><td>Lilac</td> </tr> <tr> <td>Turquoise</td><td>Peach</td><td>Opal</td> </tr> <tr> <td>Champagne</td> </tr> </table></pre> |
| XQuery |
| <pre>declare variable \$seq external; <table>{ forseq \$w in \$seq/doc/* tumbling window start position \$x when fn:true() end position \$y when \$y - \$x = 2 return <tr> {for \$i in \$w return <td>{data(\$i)}</td> } </tr> }</table></pre> |
| PositionalGrouping/arrange_rows.xq |
| Comments |

| Level Numbers |
|--|
| Convert a flat xml file with hierarchy numbers to a hierarchy xml file |
| Input |
| <pre><data> <gedcom level="0"/> <indi level="1"/> <name level="2"/> <first level="3">Michael</first> <last level="3">Kay</last> <email level="2">mike@saxonica.com</email> <indi level="1"/> <name level="2"/> <first level="3">Norm</first> <last level="3">Walsh</last> <email level="2">norm@nwalsh.com</email></pre> |

| |
|--|
| </data> |
| Output |
| <pre> <gedcom> <indi> <name> <first>Michael</first> <last>Kay</last> </name> <email>mike@saxonica.com</email> </indi> <indi> <name> <first>Norm</first> <last>Walsh</last> </name> <email>norm@nwalsh.com</email> </indi> </gedcom> </pre> |
| XQuery |
| <pre> declare variable \$input external; declare function local:group(\$seq as element()*, \$level as xs:integer) as element(){ forseq \$w in \$seq tumbling window start position \$x when \$seq[\$x]/@level eq \$level end when newstart return element {node-name(\$w[1])} {local:group(\$w, \$level+1)} }; local:group(\$input,0) </pre> <p>PositionalGrouping/level_numbers.xq</p> |
| Comments |

4. Financial Data

4.1 Transactions

One of the most important areas for using XML as an exchange format is the business sector. This is also reflected in the number of emerging standards in this area (see <http://xml.coverpages.org/xmlApplications.html>).

As this scenario has a lot in common with the gates uses cases, we concentrate just on a few examples. As the example data we use a very simplified version of cXML (<http://www.cxml.org/>) mixed with time events.

cXML Message:

```

<time date="2006-01-01T00:00:00-00:00"/>

<OrderRequest orderID="OID01" date="2006-01-01T10:00:00-00:00" type="new"
total="1100" billTo="ACME1" shipTo="ACME1">
  <Item partID="ID1" quantity="10" unitPrice="100"/>

```

```

    <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>

<OrderRequest orderID="OID02" date="2006-01-01T11:00:00-00:00" type="new"
total="100" billTo="ACME2">
    <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>

<ConfirmationRequest orderID="OID02" status="reject" date="2006-01-01T18:00:00-
00:00" confirmID="C1"/>

<time date="2006-01-02T00:00:00-00:00"/>

<ConfirmationRequest orderID="OID01" status="accept" date="2006-01-02T08:00:00-
00:00" confirmID="C1"/>

<OrderRequest orderID="OID03" date="2006-01-02T14:00:00-00:00" type="new"
total="10000" billTo="ACME1" shipTo="ACME1">
    <Item partID="ID3" quantity="100" unitPrice="100"/>
</OrderRequest>

<ConfirmationRequest orderID="OID03" status="accept" date="2006-01-02T16:00:00-
00:00" confirmID="C1"/>

<time date="2006-01-03T00:00:00-00:00"/>
<time date="2006-01-04T00:00:00-00:00"/>
<time date="2006-01-05T00:00:00-00:00"/>

<ShipNotice orderID="OID01" date="2006-01-05T08:00:00-00:00" />
<ShipNotice orderID="OID03" date="2006-01-05T09:00:00-00:00" />

<time date="2006-01-06T00:00:00-00:00"/>

<OrderRequest orderID="OID04" date="2006-01-06T08:00:00-00:00" type="new"
total="100" billTo="ACME2">
    <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>

<time date="2006-01-07T00:00:00-00:00"/>

```

4.1.1 List all open transactions

| Open transactions |
|--|
| Open transactions |
| Description |
| At the end of a day, list all open transactions. Open transactions include all orders that have been confirmed but not yet delivered. |
| Output |
| <pre> <result> <openRequests endOfDay="2006-01-01T18:00:00-00:00"> <OrderRequest orderID="OID01" date="2006-01-01T10:00:00-00:00" type="new" total="1100" billTo="ACME1" shipTo="ACME1"> <Item partID="ID1" quantity="10" unitPrice="100"/> <Item partID="ID2" quantity="10" unitPrice="10"/> </OrderRequest> </openRequests> <openRequests endOfDay="2006-01-02T16:00:00-00:00"> <OrderRequest orderID="OID01" date="2006-01-01T10:00:00-00:00" </pre> |

```

type="new" total="1100" billTo="ACME1" shipTo="ACME1">
  <Item partID="ID1" quantity="10" unitPrice="100"/>
  <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>
<OrderRequest orderID="OID03" date="2006-01-02T14:00:00-00:00"
type="new" total="10000" billTo="ACME1" shipTo="ACME1">
  <Item partID="ID3" quantity="100" unitPrice="100"/>
</OrderRequest>
</openRequests>
<openRequests endOfDay="2006-01-03T00:00:00-00:00">
  <OrderRequest orderID="OID01" date="2006-01-01T10:00:00-00:00"
type="new" total="1100" billTo="ACME1" shipTo="ACME1">
  <Item partID="ID1" quantity="10" unitPrice="100"/>
  <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>
<OrderRequest orderID="OID03" date="2006-01-02T14:00:00-00:00"
type="new" total="10000" billTo="ACME1" shipTo="ACME1">
  <Item partID="ID3" quantity="100" unitPrice="100"/>
</OrderRequest>
</openRequests>
<openRequests endOfDay="2006-01-04T00:00:00-00:00">
  <OrderRequest orderID="OID01" date="2006-01-01T10:00:00-00:00"
type="new" total="1100" billTo="ACME1" shipTo="ACME1">
  <Item partID="ID1" quantity="10" unitPrice="100"/>
  <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>
<OrderRequest orderID="OID03" date="2006-01-02T14:00:00-00:00"
type="new" total="10000" billTo="ACME1" shipTo="ACME1">
  <Item partID="ID3" quantity="100" unitPrice="100"/>
</OrderRequest>
</openRequests>
<openRequests endOfDay="2006-01-05T09:00:00-00:00"/>
<openRequests endOfDay="2006-01-06T08:00:00-00:00">
  <OrderRequest orderID="OID04" date="2006-01-06T08:00:00-00:00"
type="new" total="100" billTo="ACME2">
  <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>
</openRequests>
<openRequests endOfDay="2006-01-06T08:00:00-00:00">
  <OrderRequest orderID="OID04" date="2006-01-06T08:00:00-00:00"
type="new" total="100" billTo="ACME2">
  <Item partID="ID2" quantity="10" unitPrice="10"/>
</OrderRequest>
</openRequests>
</result>

```

XQuery

```

declare variable $seq external;
<result>{
forseq $w in $seq/sequence/* landmark window
  start position $wSPos when $wSPos eq 1
  end curItem $wECur, prevItem $wEPprev, position $wEPos
  when day-from-date(xs:dateTime ($wECur/@date)) ne
    day-from-date(xs:dateTime ($wEPprev/@date))
return
<openRequests endOfDay="{xs:dateTime($wEPprev/@date)}">{
forseq $openWindows in $w sliding window
  start curItem $oSCur when string(node-name($oSCur)) eq "OrderRequest"
  end curItem $OECur, position $OEPos when $OECur/@orderID eq $oSCur/@orderID
and

```

```

    (( string(node-name($oECur)) eq "ConfirmationRequest" and $oECur/@status eq
"reject") or string(node-name($oECur)) eq "ShipNotice")
return
if ($wEPos eq $oEPos)
then $openWindows[1]
else ()
}
</openRequests>
}</result>

```

Financial/ Q1_open_transactions.xq

Comments

4.1.2 Most valuable Customers per day

Most valuable Customers per day

At the end of a day list the most valuable customers

Output

```

<result>
  <mostValuableCustomer endOfDay="2006-01-01T00:00:00-00:00">
    <amount company="ACME1">1100</amount>
  </mostValuableCustomer>
  <mostValuableCustomer endOfDay="2006-01-02T00:00:00-00:00">
    <amount company="ACME1">10000</amount>
  </mostValuableCustomer>
  <mostValuableCustomer endOfDay="2006-01-03T00:00:00-00:00"/>
  <mostValuableCustomer endOfDay="2006-01-04T00:00:00-00:00"/>
  <mostValuableCustomer endOfDay="2006-01-05T00:00:00-00:00"/>
  <mostValuableCustomer endOfDay="2006-01-06T00:00:00-00:00">
    <amount company="ACME2">100</amount>
  </mostValuableCustomer>
  <mostValuableCustomer endOfDay="2006-01-07T00:00:00-00:00"/>
</result>

```

XQuery

```

declare variable $seq external;
<result>{
forseq $w in $seq/sequence/* sliding window
  start curItem $cur, prevItem $prev when day-from-date(xs:dateTime
    ($cur/@date)) ne day-from-date(xs:dateTime ($prev/@date)) or empty($prev)
  end when newstart
return
<mostValuableCustomer endOfDay="{xs:dateTime($cur/@date)}">{
let $companies :=
  for $x in distinct-values($w/@billTo )
  return
  <amount company="{ $x }">{sum($w[/@billTo eq $x]/@total)}</amount>
let $max := max($companies)
for $company in $companies
where $company eq xs:untypedAtomic($max)
return $company
}
</mostValuableCustomer>
}</result>

```

Financial/ Q2_most_valuable_customer.xq

Comments

4.1.3 Transaction time

| Transaction time |
|--|
| Calculate the time needed to process an order from the order request to the shipping |
| Output |
| <pre><result> <timeToShip orderID="OID01">P3DT22H</timeToShip> <timeToShip orderID="OID03">P2DT19H</timeToShip> </result></pre> |
| XQuery |
| <pre>declare variable \$seq external; <result>{ forseq \$w in \$seq/sequence/* sliding window start curItem \$s when string(node-name(\$s)) eq "OrderRequest" end curItem \$e when \$e/@orderID eq \$s/@orderID and ((string(node-name(\$e)) eq "ConfirmationRequest" and \$e/@status eq "reject") or string(node-name(\$e)) eq "ShipNotice") where string(node-name(\$e)) eq "ShipNotice" return <timeToShip orderID="{ \$s/@orderID}">{xs:dateTime(\$e/@date) - xs:dateTime(\$s/@date) }</timeToShip> }</result></pre> |
| Financial/ Q3_transaction_time.xq |
| Comments |
| |

4.1.4 Ship together

| Ship together |
|--|
| Calculates at the moment of the ship notification if an open request exists which can be shipped to the same address |
| Output |
| <pre><result> <bundleWith orderId="OID01"> <OrderRequest orderID="OID03" date="2006-01-02T14:00:00-00:00" type="new" total="10000" billTo="ACME1" shipTo="ACME1"> <Item partID="ID3" quantity="100" unitPrice="100"/> </OrderRequest> </bundleWith> <bundleWith orderId="OID03"/> </result></pre> |
| XQuery |
| <pre>declare variable \$seq external; <result>{ forseq \$w in \$seq/sequence/* sliding window start prevItem \$wSPrev when string(node-name(\$wSPrev)) eq "OrderRequest" end nextItem \$wENext when \$wENext/@orderID eq \$wSPrev/@orderID and ((string(node-name(\$wENext)) eq "ConfirmationRequest" and \$wENext/@status eq "reject") or string(node-name(\$wENext)) eq "ShipNotice") where string(node-name(\$wENext)) eq "ShipNotice" return <bundleWith orderId="{ \$wSPrev/@orderID}">{ forseq \$bundle in \$w sliding window start curItem \$bSCur when string(node-name(\$bSCur)) eq "OrderRequest" and \$bSCur/@shipTo eq \$wSPrev/@shipTo end curItem \$bECur, nextItem \$bENext when \$bECur/@orderID eq \$bSCur/@orderID and ((string(node-name(\$bECur)) eq "ConfirmationRequest" and</pre> |

| |
|---|
| <pre> \$bECur/@status eq "reject") or string(node-name(\$bECur)) eq "ShipNotice") where empty(\$bENext) return \$bSCur }</bundleWith> }</result> Financial/ Q4_ship_together.xq </pre> |
| Comments |
| |

4.2 Stock dealing (Algorithmic trading)

This category covers use cases for algorithmic trading. Most of the basic methods are quite simple, like building moving averages, exponential smoothing etc. and are already covered by the Toolbox use cases. Here we only concentrate on two typical tasks: Finding arbitrage possibilities and stock trends.

4.2.1 Stock Trend

| |
|---|
| Stock Trend |
| Many ticker sites indicate how the stock is behaving over time, if the stock is going up, strong up, down etc. This Query calculates such a trend with linear regression (least square) for a 5 min window |
| Input |
| <pre> <tick time="2006-01-01T00:01:00-00:00" price="10" type="share" tick="yhoo" /> <tick time="2006-01-01T00:02:00-00:00" price="11" type="share" tick="yhoo" /> <tick time="2006-01-01T00:03:00-00:00" price="12" type="share" tick="yhoo" /> <tick time="2006-01-01T00:04:00-00:00" price="14" type="share" tick="yhoo" /> <tick time="2006-01-01T00:05:00-00:00" price="16" type="share" tick="yhoo" /> <tick time="2006-01-01T00:06:00-00:00" price="5" type="share" tick="yhoo" /> <tick time="2006-01-01T00:07:00-00:00" price="3" type="share" tick="yhoo" /> <tick time="2006-01-01T00:08:00-00:00" price="5" type="share" tick="yhoo" /> <tick time="2006-01-01T00:09:00-00:00" price="6" type="share" tick="yhoo" /> <tick time="2006-01-01T00:10:00-00:00" price="5" type="share" tick="yhoo" /> </pre> |
| Output |
| <pre> <value tick="yhoo" score="strong up" time="2006-01-01T00:05:00-00:00"/> <value tick="yhoo" score="strong down" time="2006-01-01T00:06:00-00:00"/> <value tick="yhoo" score="strong down" time="2006-01-01T00:07:00-00:00"/> <value tick="yhoo" score="strong down" time="2006-01-01T00:08:00-00:00"/> <value tick="yhoo" score="strong down" time="2006-01-01T00:09:00-00:00"/> <value tick="yhoo" score="stable" time="2006-01-01T00:10:00-00:00"/> </pre> |
| XQuery |
| <pre> declare variable \$seq external; declare function local:sumXY(\$X as xs:double*, \$Y as xs:double*) as xs:double? { if(empty(\$X)) then 0 else \$X[1] * \$Y[1] + local:sumXY(fn:remove(\$X, 1), fn:remove(\$Y, 1)) }; declare function local:regression(\$X as xs:double*, \$Y as xs:double*) { </pre> |

```

let $n := count($X)
let $sumXY := local:sumXY($X, $Y)
let $sumX := sum($X)
let $sumY := sum($Y)
let $sumXSquare := local:sumXY($X, $X)
let $slope := ($n * $sumXY - $sumX * $sumY) div ($n * $sumXSquare - $sumX *
$sumX)
let $intercept := ($sumY - $slope * $sumX) div $n
return
<regression>
  <slope> {$slope } </slope>
  <intercept>{$intercept } </intercept>
</regression>

forseq $w in $seq/seq/* sliding window
  start curItem $s when fn:true()
  force end curItem $e when xs:dateTime($e/@time) - xs:dateTime($s/@time) eq
xs:dayTimeDuration("PT4M")
let $score := local:regression((1,2,3,4,5), $w/@time)//slope
let $forecast := if($score gt 0.5) then "strong up"
  else if ($score gt 0.2) then "up"
  else if ($score lt -0.5) then "strong down"
  else if ($score lt -0.2) then "down"
  else "stable"
return <value tick="{ $s/@tick}"score=" { $forecast}" time="{ $e/@time}"/>

```

Financial/ Q5_stock_trend.xq

Comments

4.2.2 Call-Put-Parity

$$c + PV(x) = p + s$$

c = call value, PV(x) = present value of strike price x, p=put value, s =value of the underlier.

Call-Put-Parity

Every broker house runs such a program to detect arbitrage (risk free money) possibilities. This continuous query demonstrates how the call-put-parity could be implemented using FORSEQ. Also the implementation is restricted to one tick (=stock value) it is easily extendible to different ticks.

Input

```

<tick time="0" />
<tick time="1" tick="yhoo" price="20" type="share"/>
<tick time="1" tick="yhoo" price="50" type="put" strikePrice="10" expires="3"/>
<tick time="1" tick="yhoo" price="20" type="call" strikePrice="10" expires="3"/>
<tick time="2" tick="yhoo" price="25" type="share"/>
<tick time="2" tick="yhoo" price="10" type="put" strikePrice="10" expires="3"/>
<tick time="2" tick="yhoo" price="10" type="call" strikePrice="10" expires="3"/>
<tick time="2" tick="yhoo" price="40" type="put" strikePrice="60" expires="5"/>
<tick time="2" tick="yhoo" price="20" type="call" strikePrice="60" expires="5"/>
<tick time="3" tick="yhoo" price="40" type="share"/>
<tick time="3" tick="yhoo" price="10" type="put" strikePrice="10" expires="3"/>
<tick time="3" tick="yhoo" price="10" type="call" strikePrice="10" expires="3"/>
<tick time="3" tick="yhoo" price="10" type="put" strikePrice="60" expires="5"/>
<tick time="3" tick="yhoo" price="10" type="call" strikePrice="60" expires="5"/>

```

```

<tick time="4" tick="yhoo" price="55" type="share"/>
<tick time="4" tick="yhoo" price="10" type="put" strikePrice="60" expires="5"/>
<tick time="4" tick="yhoo" price="10" type="call" strikePrice="60" expires="5"/>
<tick time="5" tick="yhoo" price="50" type="share" />
<tick time="5" tick="yhoo" price="10" type="put" strikePrice="60" expires="5"/>
<tick time="5" tick="yhoo" price="5" type="call" strikePrice="60" expires="5"/>
<tick time="6" />

```

Output

```

<arbitrage time="1"/>
<arbitrage time="2">
  <successful tick="yhoo" expireTime="3" strikePrice="10">34.90049840408766
  </successful>
  <successful tick="yhoo" expireTime="5" strikePrice="60">63.22673318790159
  </successful>
</arbitrage>
<arbitrage time="3">
  <successful tick="yhoo" expireTime="3" strikePrice="10">50</successful>
  <successful tick="yhoo" expireTime="5" strikePrice="60">98.81192118960529
  </successful>
</arbitrage>
<arbitrage time="4">
  <successful tick="yhoo" expireTime="5" strikePrice="60">114.40299042452591
  </successful>
</arbitrage>
<arbitrage time="5">
  <successful tick="yhoo" expireTime="5" strikePrice="60">105
  </successful>
</arbitrage>
<arbitrage time="6"/>

```

XQuery

```

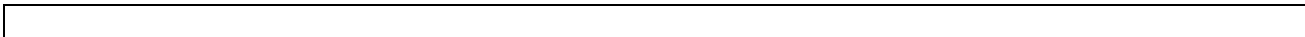
declare variable $seq external;
declare variable $interestRate := 0.01;
declare variable $treshhold := 0.5;

let $ticks := $seq/seq/tick
forseq $w in $ticks sliding window
  start curItem $sCur, prevItem $sPrev when $sCur/@time ne $sPrev/@time
  end when newstart
let $time := $sCur/@time
let $share := $w[@type eq "share"]/@price
let $tick := $w[@type eq "share"]/@tick
return
<arbitrage time="{ $time }">{
for $expires in distinct-values($w/@expires)
for $strikePrice in distinct-values($w[@expires eq $expires]/@strikePrice)
let $parityItems := $w[(@expires eq $expires and @strikePrice eq $strikePrice)]
let $call := $parityItems[@type eq "call"]/@price
let $put := $parityItems[@type eq "put"]/@price
let $constBondValue := mxq:pow(2.71828, -$interestRate * ($expires - $time))
let $callPutValue := $call + $strikePrice * $constBondValue - $put + $share
return
if($callPutValue gt $treshhold or $callPutValue lt -$treshhold)
  then <successful tick="{ $tick }" expireTime="{ $expires }"
strikePrice="{ $strikePrice }">{ $callPutValue }</successful>
  else ()
}</arbitrage>

```

Financial/ Q6_call_put_parity.xq

Comments



5. Toolbox

The toolbox use cases are the most general cases which are often needed to pre-/post-process streams. Typical tasks in this category are

- Compute the moving average.
- Compute the exponential smoothing.
- For each sensor maintain the current value.
- Stream synchronization.
- Non-blocking min/max: Return a stream of the current min/max value. Return the current max/min value for every hour.
- Outlier detection: Report if a value is higher (lower) than 150% (50%) of the last value.
- Pre-process a stream so that it is sorted by time. If an event comes with a delay of more than X seconds, than this event is dropped.
- Filter out repetitive alarms.

In the next section XQuery solutions are provided to carry out these tasks.

5.1 Moving average

| |
|--|
| 1. Moving average (based on time). |
| Calculate moving average of temperature values for N (e.g. 3) last seconds. |
| Input Delayed Events Day1 |
| <pre><stream> <event id="1" created="1" arrived="3" temp='10' /> <event id="2" created="2" arrived="3" temp='8' /> <event id="3" created="2" arrived="3" temp='6' /> <event id="4" created="0" arrived="3" temp='12' /> <event id="5" created="1" arrived="7" temp='10' /> <event id="6" created="3" arrived="7" temp='10' /> <event id="7" created="4" arrived="7" temp='10' /> <event id="8" created="5" arrived="7" temp='10' /> <event id="9" created="4" arrived="7" temp='10' /> <event id="10" created="3" arrived="7" temp='10' /> <event id="11" created="8" arrived="10" temp='10' /> <event id="12" created="10" arrived="11" temp='12' /> <event id="13" created="9" arrived="12" temp='14' /> </stream></pre> |

| |
|---|
| Output |
| 9 10 12 13 14 |
| XQuery |
| <pre> declare variable \$timesequence external; let \$MAX_DIFF := 3 forseq \$w in \$timesequence/stream/event sliding window start curItem \$s_curr, prevItem \$s_prev, position \$s_pos when (\$s_curr/@arrived ne \$s_prev/@arrived) or (\$s_pos eq 1) end nextItem \$e_next when \$e_next/@arrived - \$s_curr/@arrived gt \$MAX_DIFF return { avg(\$w/@temp) } </pre> <p>Toolbox/moving_avarage_time.xq</p> |
| Comments |
| Result is returned every time a new event is received (except first 2 events). |

| |
|--|
| 2. Moving average (based on number of events). |
| Calculate moving average of temperature values for N (e.g. 4) last events. |
| Input |
| The same as in previous example: Delayed Events Day1 |
| Output |
| 9 9 9.5 10.5 10 10 10 10 10.5 11.5 12 13 14 |
| XQuery |
| <pre> declare variable \$timesequence external; let \$MAX_SIZE := 4 forseq \$w in \$timesequence/stream/event sliding window start position \$s_pos when true() end position \$e_pos when \$e_pos - \$s_pos eq \$MAX_SIZE -1 return { avg(\$w/@temp) } </pre> <p>Toolbox/moving_avarage_size.xq</p> |
| Comments |
| Result is returned every time a new event is received (except first 3 events) |

5.2 Exponential Smoothing

| |
|---|
| 1. Single Exponential Smoothing (N last values) |
| <p><u>Single Exponential Smoothing</u>: $S_t = \alpha y_{t-1} + (1-\alpha) S_{t-1}$ $0 < \alpha < 1$, $t \geq 3$ N (e.g. 3) last values of the sequence are taken into account while smoothing current value.</p> <p>More detailed description see e.g.: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm</p> |
| Input |
| The same as in 5.1 example “1. Moving average (based on time)”. |

Output

```
8.379999999999999 8.78 8.459999999999999 10.979999999999999 10 10 10 10 10 10.6
11.62
```

XQuery 1 (take into account only 3 last values)

```
declare variable $timesequence external;
let $SMOOTH_CONST := 0.3

forseq $w in $timesequence/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq 2
return
{
  $SMOOTH_CONST * data($w[3]/@temp) + (1 - $SMOOTH_CONST) *
  ( $SMOOTH_CONST * data($w[2]/@temp) + (1 - $SMOOTH_CONST) * data($w[1]/@temp)
)
}
```

Toolbox/exponential_smoothing_1.xq

XQuery 2 (more general solution; recursion)

```
xquery version "1.0";

declare namespace f = 'test';
declare variable $timesequence external;
declare variable $SMOOTH_CONST := 0.3;
declare variable $WND_SIZE := 3;

declare function f:calc-exp
($seq as element()*, $pos as xs:integer, $res as xs:integer) as xs:integer
{
  let $fRes := if ($pos le $WND_SIZE)
  then
    f:calc-exp($seq, $pos +1, $SMOOTH_CONST * $seq[$pos]/@temp +
      (1 - $SMOOTH_CONST)* $res)
  else $res
return $fRes
};

forseq $w in $timesequence/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq $WND_SIZE -1
return
{
  f:calc-exp( $w, 2, $w[1]/@temp )
}
```

Toolbox/exponential_smoothing_2.xq

XQueryP

```
declare execution sequential;
declare variable $timesequence external;
declare variable $first := true();
declare variable $SMOOTH_CONST := 0.3;
declare variable $WND_SIZE := 3;
declare variable $res;

forseq $w in $timesequence/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq $WND_SIZE -1
return
{
```

```

set $first := true();
for $e in $w
return $WND_SIZE
{
  set $res := if ($first)
    then { set $first := false(); data($e/@temp) }
    else $SMOOTH_CONST * data($e/@temp) + (1 - $SMOOTH_CONST) * $res
};
$res
}

```

Toolbox/exponential_smoothing_xqp_1.xq

Comments

- In solution XQuery1 only the three last values are evaluated while calculating current value. It would be pretty hard to change the query and take into account 4 values instead of 3.
- Solutions XQuery2 and XQueryP are more general in that sense. It's easy to change the number of items to be evaluated by changing the variable \$WND_SIZE.
- Solution XQuery2 uses recursive function.
- Solution XQueryP uses update functionality, to keep the state of the computation.

2. Single Exponential Smoothing (all previous values)

Single Exponential Smoothing: $S_t = \alpha y_{t-1} + (1-\alpha) S_{t-1}$ $0 < \alpha < 1$, $t \geq 3$

All previous values are taken into account while smoothing current value.

More detailed description:

<http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm>

Input

The same as in 5.1 example "1. Moving average (based on time)".

Output

```

10 9.4 8.379999999999999 9.465999999999998 9.626199999999997 9.738339999999997
9.816837999999997 9.871786599999997 9.910250619999998 9.937175433999997
9.956022803799996 10.569215962659996 11.598451173861996

```

XQuery

```

declare execution sequential;
declare variable $timesequence external;
declare variable $first := true();
declare variable $SMOOTH_CONST := 0.3;
declare variable $res;

for $e in $timesequence/stream/event
return
{
  set $res := if ($first) then { set $first := false(); data($e/@temp) }
  else $SMOOTH_CONST * data($e/@temp) + (1 - $SMOOTH_CONST) * $res;
  $res
}

```

Toolbox/exponential_smoothing_xqp_1.xq

XQueryP

```

declare execution sequential;
declare variable $timesequence external;
declare variable $first := true();
declare variable $SMOOTH_CONST := 0.3;
declare variable $res;

```

```

for $e in $timesequence/stream/event
return
{
  set $res := if ($first) then { set $first := false(); data($e/@temp) }
  else $SMOOTH_CONST * data($e/@temp) + (1 - $SMOOTH_CONST) * $res;
  $res
}

```

Toolbox/exponential_smoothing_xqp_1.xq

Comments

3. Double Exponential Smoothing (N last values)

Double Exponential Smoothing:

N (e.g. 3) last values of the sequence are taken into account while smoothing current value.

$$S_t = \alpha y_t + (1-\alpha) (S_{t-1} + b_{t-1}) \quad 0 < \alpha < 1$$

$$b_t = q(S_t - S_{t-1}) + (1-q) b_{t-1} \quad 0 < q < 1$$

More detailed description:

<http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc433.htm>

Input

The same as in 5.1 example: Delayed Events Day1

Output

5.9999999999999999 6.3999999999999999 15.5999999999999996 8.6 10 10 10 10 10 10.6
14

XQuery

```

xquery version "1.0";

declare namespace f = 'test';
declare variable $timesequence external;
declare variable $SMOOTH_CONST := 0.3;
declare variable $TREND_CONST := 0.3;
declare variable $WND_SIZE := 3;

declare function f:calc-exp($seq as element(*), $pos as xs:integer,
  $res as xs:float, $trend as xs:float) as xs:float
{
  let $newRes :=
    $SMOOTH_CONST * $seq[$pos]/@temp + (1 - $SMOOTH_CONST) * ($res + $trend)

  let $newTrend :=
    $TREND_CONST * ($newRes - $res) + (1 - $TREND_CONST) * $trend

  let $fRes :=
    if ($pos le $WND_SIZE) then
      f:calc-exp($seq, $pos + 1, $newRes, $newTrend)
    else $res
  return $fRes
};

forseq $w in $timesequence/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq $WND_SIZE - 1
return

```

```
{
  f:calc-exp( $w, 2, $w[1]/@temp, $w[2]/@temp - $w[1]/@temp)
}
```

Toolbox/exponential_double_smoothing_2.xq

XQueryP

```
declare execution sequential;
declare variable $timesequence external;
declare variable $first := true(); (: change to position in for loop :)
declare variable $SMOOTH_CONST := 0.3;
declare variable $TREND_CONST := 0.3;
declare variable $WND_SIZE := 3;
declare variable $res;
declare variable $prevRes;
declare variable $trend;
declare variable $tmp;

forseq $w in $timesequence/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq $WND_SIZE -1
return
{
  set $first := true();
  for $e in $w
  return
  {
    set $tmp := if ($first)
    then
    {
      set $first := false();
      set $trend := data($w[2]/@temp) - data($w[1]/@temp);
      set $res := data($e/@temp)
    }
    else
    {
      set $prevRes := $res;
      set $res := $SMOOTH_CONST * data($e/@temp) + (1 - $SMOOTH_CONST) * ($res
+ $trend);
      set $trend := $TREND_CONST * ($res - $prevRes) + (1 - $TREND_CONST) *
$trend
    }
  };
  $res
}
```

Toolbox/exponential_double_smoothing_xqp.xq

Comments

- Both solutions (XQuery and XQueryP) allow for easy change of the number of items to be evaluated while smoothing current value (variable \$WND_SIZE).
- XQuery solution uses recursive function.
XQueryP solution uses update functionality, to keep the state of the calculation.

5.3 Current max (min) value

1. Maximum value (reported periodically)

| |
|---|
| Every hour return the maximum temperature value within that hour. Each event has attributes: ID, creation time, sensor ID and temperature value. |
| Input |
| <pre><stream> <event id="1" time="2006-01-01T01:00:00-00:00" sensor="1" temp='10' /> <event id="2" time="2006-01-01T01:30:00-00:00" sensor="1" temp='8' /> <event id="3" time="2006-01-01T02:00:00-00:00" sensor="2" temp='6' /> <event id="4" time="2006-01-01T02:30:00-00:00" sensor="2" temp='12' /> <event id="5" time="2006-01-01T03:00:00-00:00" sensor="2" temp='10' /> <event id="6" time="2006-01-01T03:00:00-00:00" sensor="1" temp='9' /> <event id="7" time="2006-01-01T03:30:00-00:00" sensor="2" temp='16' /> <event id="8" time="2006-01-01T03:30:00-00:00" sensor="2" temp='20' /> <event id="9" time="2006-01-01T03:30:00-00:00" sensor="3" temp='10' /> <event id="10" time="2006-01-01T04:00:00-00:00" sensor="1" temp='11' /> <event id="11" time="2006-01-01T04:00:00-00:00" sensor="2" temp='20' /> <event id="12" time="2006-01-01T04:00:00-00:00" sensor="3" temp='25' /> <event id="11" time="2006-01-01T04:30:00-00:00" sensor="2" temp='18' /> <event id="12" time="2006-01-01T04:30:00-00:00" sensor="3" temp='15' /> </stream></pre> |
| Output |
| <pre><res><h>1</h><max>10</max></res> <res><h>2</h><max>12</max></res> <res><h>3</h><max>20</max></res> <res><h>4</h><max>25</max></res></pre> |
| XQuery |
| <pre>declare variable \$seq external; forseq \$w in \$seq/stream/event tumbling window start curItem \$s_curr, prevItem \$s_prev, position \$s_pos when fn:hours-from-dateTime(xs:dateTime(\$s_curr/@time)) ne fn:hours-from-dateTime(xs:dateTime(\$s_prev/@time)) or (\$s_pos eq 1) end when newstart return <res> <h> { fn:hours-from-dateTime(xs:dateTime(\$s_curr/@time)) } </h> <max> { max(\$w/@temp) } </max> </res></pre> <p>Toolbox/non_blocking_max1.xq</p> |
| Comments |

| |
|---|
| 2. Maximum value per sensor (reported periodically) |
| Every hour return the maximum temperature value within that hour per sensor. |
| Input |
| The same as in 5.3 example 1. Maximum value (reported periodically). |
| Output |
| <pre><res><hour>1</hour><sen>1</sen><max>10</max></res> <res><hour>2</hour><sen>2</sen><max>12</max></res> <res><hour>3</hour><sen>2</sen><max>20</max></res> <res><hour>3</hour><sen>1</sen><max>9</max></res> <res><hour>3</hour><sen>3</sen><max>10</max></res> <res><hour>4</hour><sen>1</sen><max>11</max></res> <res><hour>4</hour><sen>2</sen><max>20</max></res> <res><hour>4</hour><sen>3</sen><max>25</max></res></pre> |
| XQuery |

```

declare variable $seq external;

forseq $w in $seq/stream/event tumbling window
  start curItem $s_curr, prevItem $s_prev, position $s_pos when
    fn:hours-from-dateTime(xs:dateTime($s_curr/@time)) ne
    fn:hours-from-dateTime(xs:dateTime($s_prev/@time)) or ($s_pos eq 1)
  end when newstart
return
  for $s in distinct-values($w/@sensor)
  return
    <res>
      <hour>{fn:hours-from-dateTime(xs:dateTime($s_curr/@time))}</hour>
      <sen>{$s}</sen>
      <max> { max($w[@sensor eq $s]/@temp) } </max>
    </res>

```

Toolbox/non_blocking_max2.xq

Comments

3. Current maximum value (within specified duration).

For every event with a new time value return time of the event and current maximum value within last hour. E.g. if there are 3 events at the same second, the result will be returned after the 3rd event is received.

Input

The same as in 5.3 example 1. Maximum value (reported periodically).

Output

```

<res time="2006-01-01T02:00:00-00:00"><max>10</max></res>
<res time="2006-01-01T02:30:00-00:00"><max>12</max></res>
<res time="2006-01-01T03:00:00-00:00"><max>12</max></res>
<res time="2006-01-01T03:30:00-00:00"><max>20</max></res>
<res time="2006-01-01T04:00:00-00:00"><max>25</max></res>

```

XQuery

```

declare variable $seq external;

forseq $w in $seq/stream/event sliding window
  start curItem $s_curr, prevItem $s_prev, position $s_pos when
    xs:dateTime($s_curr/@time) ne xs:dateTime($s_prev/@time) or
    ($s_pos eq 1)
  force end curItem $e_curr, nextItem $e_next when
    xs:dateTime($e_next/@time) - xs:dateTime($s_curr/@time) gt
    xs:dayTimeDuration('PT1H')
return
  <res>
    { $e_curr/@time }
    <max>{ max($w/@temp) }</max>
  </res>

```

Toolbox/non_blocking_max_curr2.xq

Comments

- No results returned for the events in the first hour.
- No results returned for the events in the last hour: namely, starting event with time value 2006-01-01T03:30:00-00:00. That is because force end requires end condition to be satisfied (event with time value later than 2006-01-01T04:30:00-00:00 is needed).

| 4. Current maximum value (within specified duration). |
|--|
| <p>Previous query is extended with additional requirement: return only the most up to date values (if there was an event delay).</p> <p>Each event has attributes: ID, creation time, sensor ID and temperature value.</p> |
| Input |
| <pre><stream> <event id="1" time="2006-01-01T01:00:00-00:00" sensor="1" temp='10' /> <event id="2" time="2006-01-01T02:01:00-00:00" sensor="1" temp='12' /> <event id="3" time="2006-01-01T02:10:00-00:00" sensor="2" temp='6' /> <event id="4" time="2006-01-01T02:30:00-00:00" sensor="2" temp='8' /> <event id="5" time="2006-01-01T03:45:00-00:00" sensor="2" temp='30' /> <event id="6" time="2006-01-01T03:50:00-00:00" sensor="1" temp='25' /> <event id="7" time="2006-01-01T04:46:00-00:00" sensor="1" temp='11' /> </stream></pre> |
| Output |
| <pre><res time="2006-01-01T01:00:00-00:00"><max>10</max></res> <res time="2006-01-01T02:30:00-00:00"><max>8</max></res> <res time="2006-01-01T03:50:00-00:00"><max>30</max></res></pre> |
| XQuery |
| <pre>declare variable \$seq external; forseq \$resWnd in (forseq \$w in \$seq/stream/event sliding window start curItem \$s_curr, prevItem \$s_prev, position \$s_pos when xs:dateTime(\$s_curr/@time) ne xs:dateTime(\$s_prev/@time) or (\$s_pos eq 1) force end curItem \$e_curr, nextItem \$e_next when xs:dateTime(\$e_next/@time) - xs:dateTime(\$s_curr/@time) gt xs:dayTimeDuration('PT1H') return <res> { \$e_curr/@time } <max>{ max(\$w/@temp) } </max> </res>) tumbling window start curItem \$rs_curr, prevItem \$rs_prev, position \$rs_pos when \$rs_curr/@time ne \$rs_prev/@time or (\$rs_pos eq 1) end curItem \$re_curr, nextItem \$re_next, position \$re_pos when \$re_curr/@time ne \$re_next/@time return \$resWnd[last()]</pre> <p><small>Toolbox/non_blocking_max_curr3.xq</small></p> |
| Comments |
| <ul style="list-style-type: none"> - Interesting example. The inner forseq query is exactly as in previous example. So, the same comments are relevant. - The result sequence produced by the inner query is used in another forseq and further adjusted: the older not so relevant results are filtered out. - E.g. at time 2006-01-01T03:45:00-00:00 event with id 5 is received (after more than 1h since last event) and 3 windows w1[event id's: 2,3,4], w2[event id's: 3,4], w3[event id's: 4] are closed and evaluated. Since the result of window w3 will be the most recent (it does not have as much old events as other windows) the results of the w1 and w2 are filtered out in the outer forseq query. |

| |
|--|
| 5. Current maximum value (within specified duration). |
| Every time a new event is received immediately return current maximum value within last hour. |
| Input |
| The same as in 5.3 example 4. Current maximum value (within specified duration). |
| Output |
| <pre><res>10</res> <res>12</res> <res>12</res> <res>12</res> <res>30</res> <res>30</res> <res>25</res></pre> |
| XQuery |
| <pre>declare variable \$seq external; forseq \$w in \$seq/stream/event landmark window start position \$s_pos when \$s_pos eq 1 force end curItem \$e_curr when true() let \$filtered := \$w[@time > xs:dateTime(\$e_curr/@time)- xs:dayTimeDuration('PT1H') and @time <= xs:dateTime(\$e_curr/@time)] return <res>{ max(\$filtered/@temp) }</res></pre> <p>Toolbox/non_blocking_max_curr4.xq</p> |
| Comments |
| |

5.4 Current sensor value

| |
|---|
| 1. Current sensor value. |
| For each sensor maintain the current temperature value. Every time new event from any sensor is received, output the most recent values (received some time ago) of all sensors. Each event has attributes: ID, timestamp, sensor ID and temperature value. |
| Input |
| <pre><stream> <event id="1" time="1" sensor="1" temp='10' /> <event id="2" time="2" sensor="1" temp='8' /> <event id="3" time="2" sensor="2" temp='6' /> <event id="4" time="3" sensor="2" temp='12' /> <event id="5" time="4" sensor="3" temp='10' /> <event id="6" time="5" sensor="1" temp='9' /> <event id="7" time="5" sensor="2" temp='10' /> <event id="8" time="6" sensor="3" temp='11' /> </stream></pre> |
| Output |
| <pre><state><sensor sensor="1" time="1" temp="10"></sensor></state> <state><sensor sensor="1" time="2" temp="8"></sensor></state> <state><sensor sensor="1" time="2" temp="8"></sensor> <sensor sensor="2" time="2" temp="6"></sensor></state> <state><sensor sensor="1" time="2" temp="8"></sensor> <sensor sensor="2" time="3" temp="12"></sensor> </state></pre> |

```

<state><sensor sensor="1" time="2" temp="8"></sensor>
  <sensor sensor="2" time="3" temp="12"></sensor>
  <sensor sensor="3" time="4" temp="10"></sensor>
</state>
<state><sensor sensor="1" time="5" temp="9"></sensor>
  <sensor sensor="2" time="3" temp="12"></sensor>
  <sensor sensor="3" time="4" temp="10"></sensor>
</state>
<state><sensor sensor="1" time="5" temp="9"></sensor>
  <sensor sensor="2" time="5" temp="10"></sensor>
  <sensor sensor="3" time="4" temp="10"></sensor>
</state>
<state><sensor sensor="1" time="5" temp="9"></sensor>
  <sensor sensor="2" time="5" temp="10"></sensor>
  <sensor sensor="3" time="6" temp="11"></sensor>
</state>

```

XQuery

```

declare variable $timesequence external;

forseq $w in $timesequence/stream/event landmark window
  start position $s_pos when $s_pos eq 1
  force end when true()
return
  <state>
  {
    for $s in distinct-values($w/@sensor)
    return
      <sensor> {
        let $val:= $w[@sensor eq $s][last()]
        return ($val/@sensor, $val/@time, $val/@temp)
      } </sensor>
  }
</state>
Toolbox/maintain_current_value.xq

```

Comments

To drop inactive sensors a combination with a time-based sliding window might be useful.

2. Current sensor value (within specified duration).

For each sensor maintain the current temperature value.

Previous query is extended with additional requirement: sensor values are maintained in a time-based sliding window (e.g. of 3 seconds)

Input

The same as in previous example.

Output

```

<state>
  <sensor sensor="1" time="2" temp="8"></sensor>
  <sensor sensor="2" time="3" temp="12"></sensor>
  <sensor sensor="3" time="4" temp="10"></sensor>
</state>
<state>
  <sensor sensor="2" time="5" temp="10"></sensor>
  <sensor sensor="3" time="4" temp="10"></sensor>
  <sensor sensor="1" time="5" temp="9"></sensor>
</state>

```

XQuery

```

declare variable $timesequence external;
let $MAX_DIFF := 3

```

```

forseq $w in $timesequence/stream/event sliding window
start prevItem $s_prev, curItem $s_curr, position $s_pos when
  s_prev/@time ne $s_curr/@time
force end nextItem $e_next when
  $e_next/@time - $s_curr/@time eq $MAX_DIFF
return
<state>
{
  for $s in distinct-values($w/@sensor)
  return
    <sensor> {
      let $val:= $w[@sensor eq $s][last()]
      return ($val/@sensor, $val/@time, $val/@temp)
    } </sensor>
}
</state>

```

Toolbox/maintain_current_value_in_wnd.xq

Comments

Results are not returned for the events in the first 3 seconds (because of the sliding window).

5.5 Stream Sorting

Sort by given parameter

Pre-process a stream so that it is sorted by time of event creation.

Each event has attributes: ID, timestamp of event creation, timestamp of event arrival and temperature value. If an event comes with a delay of more than X (e.g. 3) seconds, this event is dropped. Return sorted event creation timestamps.

Input (Delayed Events Day1)

```

<stream>
  <event id="1" created="1" arrived="3" temp='10' />
  <event id="2" created="2" arrived="3" temp='8' />
  <event id="3" created="2" arrived="3" temp='6' />
  <event id="4" created="0" arrived="3" temp='12' />
  <event id="5" created="1" arrived="7" temp='10' />
  <event id="6" created="3" arrived="7" temp='10' />
  <event id="7" created="4" arrived="7" temp='10' />
  <event id="8" created="5" arrived="7" temp='10' />
  <event id="9" created="4" arrived="7" temp='10' />
  <event id="10" created="3" arrived="7" temp='10' />
  <event id="11" created="8" arrived="10" temp='10' />
  <event id="12" created="10" arrived="11" temp='12' />
  <event id="13" created="9" arrived="12" temp='14' />
</stream>

```

Output

0 1 2 2 4 4 5 8 9 10

XQuery

```

declare variable $timesequence external;

let $MAX_DIFF := 3
forseq $w in $timesequence/stream/event tumbling window
  start curItem $s when $s/@arrived - $s/@created le $MAX_DIFF
  end nextItem $e when $e/@arrived - $s/@arrived gt $MAX_DIFF or $e/@arrived -
  $e/@created gt $MAX_DIFF
return

```

```
{
  for $i in $w
  order by xs:integer($i/@created) ascending
  return xs:integer($i/@created)
}
```

Toolbox/sort_by_time.xq

Comments

5.6 Filter out repetitive alarms

Filter out repetitive events.

E.g. several following alarm events.

Each event has only one attribute - event ID.

Input

```
<stream>
  <event id="1"/>
  <event id="2"/>
  <event id="2"/>
  <event id="2"/>
  <event id="3"/>
  <event id="1"/>
  <event id="1"/>
  <event id="1"/>
  <event id="1"/>
  <event id="3"/>
  <event id="2"/>
  <event id="2"/>
  <event id="1"/>
  <event id="1"/>
  <event id="3"/>
</stream>
```

Output (XQuery 1)

```
<event id="1"></event>
<event id="2"></event>
<event id="3"></event>
<event id="1"></event>
<event id="3"></event>
<event id="2"></event>
<event id="1"></event>
<event id="3"></event>
```

Output (XQuery 2)

```
<event id="2"></event>
<event id="3"></event>
<event id="1"></event>
<event id="3"></event>
<event id="2"></event>
<event id="1"></event>
<event id="3"></event>
```

XQuery 1

```
declare variable $timesequence external;

forseq $w in $timesequence/stream/event tumbling window
  start curItem $s when true()
```

```

    end nextItem $e when $s/@id ne $e/@id
return
$w[1]
Toolbox/filter_repetitive_events2.xq

```

XQuery 2

```

declare variable $timesequence external;

forseq $w in $timesequence/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq 1
return
  if ( $w[1]/@id eq $w[2]/@id )
  then ()
  else $w[2]

Toolbox/filter_repetitive_events.xq

```

Comments

- Solution XQuery 1 collects all items with the same id into the window and returns only the first item from that window.
- Solution XQuery 2 uses window with two items for comparison. This solution will not return the first item of the sequence (see. Introduction section for more details).

5.7 Stream synchronization

1. Synchronize two streams.

The result sequence (stream) will contain all items from both input sequences sorted by event timestamp. Each event has attributes: timestamp and sequence ID.

Input

```

SEQ1:
<stream>
  <event time="-1" s="1"/>
  <event time="1" s="1"/>
  <event time="2" s="1"/>
  <event time="3" s="1"/>
  <event time="5" s="1"/>
  <event time="5" s="1"/>
  <event time="9" s="1"/>
</stream>

```

```

SEQ2:
<stream>
  <event time="1" s="2"/>
  <event time="2" s="2"/>
  <event time="2" s="2"/>
  <event time="4" s="2"/>
  <event time="4" s="2"/>
  <event time="6" s="2"/>
  <event time="7" s="2"/>
  <event time="8" s="2"/>
  <event time="10" s="2"/>
</stream>

```

Output

```

<event time="1" s="2"></event>
<event time="1" s="1"></event>
<event time="2" s="2"></event>

```

```

<event time="2" s="2"></event>
<event time="2" s="1"></event>
<event time="3" s="1"></event>
<event time="4" s="2"></event>
<event time="4" s="2"></event>
<event time="5" s="1"></event>
<event time="5" s="1"></event>
<event time="6" s="2"></event>
<event time="7" s="2"></event>
<event time="8" s="2"></event>
<event time="9" s="1"></event>

```

XQuery

```

declare variable $seq1 external;
declare variable $seq2 external;

forseq $w in $seq1/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq 1
return
  (for $e in $seq2/stream/event
   return
     if ( xs:integer($w[1]/@time) < xs:integer($e/@time) and
         xs:integer($e/@time) <= xs:integer($w[2]/@time) )
     then $e
     else (), $w[2])

```

Toolbox/synchronization1.xq

Comments

Events from SEQ2 with higher values than the last event from SEQ1 are not included into result (if the sequences are infinite that should not happen). This problem is solved in the next query.

2. Synchronize two streams (special case with finite streams).

This solution solves the problem of including all events into the result sequence (more detailed description in the previous query comment part) for finite sequences.

Each event has attributes: ID, timestamp and sequence ID.

Input

```

SEQ1:
<stream>
  <event id="1" time="-1" s="1"/>
  <event id="2" time="1" s="1"/>
  <event id="3" time="1" s="1"/>
  <event id="4" time="7" s="1"/>
</stream>

```

```

SEQ2:
<stream>
  <event id="1" time="1" s="2"/>
  <event id="2" time="1" s="2"/>
  <event id="3" time="3" s="2"/>
  <event id="4" time="5" s="2"/>
  <event id="5" time="7" s="2"/>
  <event id="6" time="8" s="2"/>
</stream>

```

Output

```

<event id="1" time="1" s="2"></event>
<event id="2" time="1" s="2"></event>
<event id="2" time="1" s="1"></event>

```

```

<event id="3" time="1" s="1"></event>
<event id="3" time="3" s="2"></event>
<event id="4" time="5" s="2"></event>
<event id="5" time="7" s="2"></event>
<event id="4" time="7" s="1"></event>
<event id="6" time="8" s="2"></event>

```

XQuery

```

declare variable $seq1 external;
declare variable $seq2 external;

forseq $w in $seq1/stream/event sliding window
  start position $s_pos when true()
  force end position $e_pos when $e_pos - $s_pos eq 1
return
  (for $e in $seq2/stream/event
  return
    if ( xs:integer($w[1]/@time) lt xs:integer($e/@time) and
    xs:integer($e/@time) le xs:integer($w[2]/@time) )
    then $e
    else (), $w[2],
      let $res :=
        if ( $seq1/stream/event/@id[last()] eq $w/@id[last()] )
        then
          for $l in $seq2/stream/event
          return
            if ( xs:integer($l/@time) gt
            xs:integer($w[2]/@time) )
            then $l
            else ()
          else ()
          return $res
    )
  Toolbox/synchronization2.xq

```

Comments

3. Synchronize stream with time.

Every day at 8 o'clock report all events which happened in last 24 hours. Return warning message, if there were no events at all.

Time sequence (*TIME_SEQ*) event has only one attribute time value (time events should be produced every hour; In the example most of the unimportant time events are skipped).

Event sequence (*EVENT_SEQ*) event has two attributes: ID and time value.

Input

```

TIME SEQ:
<stream>
  <event time="2006-01-01T01:00:00-00:00" />
  <event time="2006-01-01T08:00:00-00:00" />
  <event time="2006-01-01T10:00:00-00:00" />
  <event time="2006-01-02T04:00:00-00:00" />
  <event time="2006-01-02T08:00:00-00:00" />
  <event time="2006-01-02T12:00:00-00:00" />
  <event time="2006-01-03T01:00:00-00:00" />
  <event time="2006-01-03T08:00:00-00:00" />
  <event time="2006-01-04T08:00:00-00:00" />
  <event time="2006-01-05T08:00:00-00:00" />
</stream>

```

EVENT SEQ:

```
<stream>
  <event id="1" time="2006-01-01T01:00:00-00:00"/>
  <event id="2" time="2006-01-02T05:00:00-00:00"/>
  <event id="3" time="2006-01-02T07:00:00-00:00"/>
  <event id="4" time="2006-01-02T08:00:00-00:00"/>
  <event id="5" time="2006-01-02T11:00:00-00:00"/>
  <event id="6" time="2006-01-02T12:00:00-00:00"/>
  <event id="7" time="2006-01-03T01:00:00-00:00"/>
  <event id="8" time="2006-01-05T03:00:00-00:00"/>
</stream>
```

Output

```
<res>
  <event id="2" time="2006-01-02T05:00:00-00:00"></event>
  <event id="3" time="2006-01-02T07:00:00-00:00"></event>
</res>
<res>
  <event id="4" time="2006-01-02T08:00:00-00:00"></event>
  <event id="5" time="2006-01-02T11:00:00-00:00"></event>
  <event id="6" time="2006-01-02T12:00:00-00:00"></event>
  <event id="7" time="2006-01-03T01:00:00-00:00"></event>
</res>
<warning>No data from 2006-01-03T08:00:00-00:00 to 2006-01-04T08:00:00-00:00
</warning>
<res>
  <event id="8" time="2006-01-05T03:00:00-00:00"></event>
</res>
```

XQuery

```
declare variable $timeSeq external;
declare variable $seq external;

forseq $w in $timeSeq/stream/event sliding window
  start curItem $s_time, position $s_tpos when
  xs:time(xs:dateTime($s_time/@time)) eq xs:time('08:00:00-00:00')
  force end curItem $e_time, position $e_tpos when
  xs:time(xs:dateTime($e_time/@time)) eq xs:time('08:00:00-00:00')
  and ($s_tpos ne $e_tpos)
return
  let $val :=
  forseq $valW in $seq/stream/event tumbling window
    start curItem $s_val when
      xs:dateTime($s_val/@time) ge xs:dateTime($s_time/@time) and
      xs:dateTime($s_val/@time) lt xs:dateTime($e_time/@time)
    end nextItem $e_val when
      xs:dateTime($e_val/@time) ge xs:dateTime($e_time/@time)
  return $valW

let $res := if ( empty($val) )
  then <warning>No data from{xs:dateTime($s_time/@time)}
to{xs:dateTime($e_time/@time)} </warning>
  else <res> {$val} </res>
return $res

Toolbox/synchronization2.xq
```

Comments

5.8 Outlier detection

| 1. Outlier detection (explosive outlier). |
|---|
| <p>Sensors are periodically reporting temperature values. Outlier is detected and reported, if current value of the sensor is two times higher (lower) than a previous one. Each event has attributes: ID, timestamp, sensor ID and temperature value.</p> |
| Input |
| <pre><stream> <event id="1" time="1" sensor="1" temp='10' /> <event id="2" time="2" sensor="1" temp='9' /> <event id="3" time="2" sensor="2" temp='10' /> <event id="4" time="3" sensor="2" temp='35' /> <event id="5" time="4" sensor="2" temp='25' /> <event id="6" time="5" sensor="2" temp='22' /> <event id="7" time="6" sensor="2" temp='5' /> <event id="8" time="7" sensor="2" temp='7' /> <event id="9" time="8" sensor="2" temp='10' /> <event id="10" time="9" sensor="2" temp='11' /> <event id="11" time="10" sensor="1" temp='20' /> <event id="12" time="11" sensor="1" temp='25' /> </stream></pre> |
| Output |
| <pre><alarm>Outlier detected. Event id: 4</alarm> <alarm>Outlier detected. Event id: 7</alarm> <alarm>Outlier detected. Event id: 11</alarm></pre> |
| XQuery |
| <pre>declare variable \$seq external; forseq \$w in \$seq/stream/event sliding window start curItem \$s_curr, position \$s_pos when true() force end curItem \$e_curr, position \$e_pos when \$s_curr/@sensor eq \$e_curr/@sensor and \$e_pos ne \$s_pos return if (\$s_curr/@temp * 2 lt xs:integer(\$e_curr/@temp) or xs:integer(\$s_curr/@temp) gt \$e_curr/@temp * 2) then <alarm>Outlier detected. Event id:{data(\$e_curr/@id)}</alarm> else () Toolbox/outlier_detection.xq</pre> |
| Comments |
| |

| 2. Outlier detection (incremental outlier). |
|--|
| <p>Sensors are periodically reporting temperature values. In previous example outlier was detected if two subsequent events had significant temperature value difference (e.g. 100%). This example handles the case when sensor starts to report incorrect values by increasing/decreasing them constantly. In this case outlier is detected, if minimum and maximum temperature values reported by sensor in N (e.g. 3) subsequent events have significant difference (e.g. 100%). Each event has attributes: ID, timestamp, sensor ID and temperature value.</p> |
| Input |
| <pre><stream> <event id="1" time="1" sensor="1" temp='10' /> <event id="2" time="2" sensor="1" temp='9' /> <event id="3" time="2" sensor="2" temp='10' /></pre> |

```

<event id="4" time="3" sensor="2" temp='15' />
<event id="5" time="4" sensor="2" temp='21' />
<event id="6" time="5" sensor="2" temp='22' />
<event id="7" time="6" sensor="2" temp='20' />
<event id="8" time="7" sensor="2" temp='18' />
<event id="9" time="8" sensor="2" temp='16' />
<event id="10" time="9" sensor="2" temp='14' />
<event id="11" time="10" sensor="1" temp='21' />
<event id="12" time="11" sensor="1" temp='25' />
</stream>

```

Output

```

<alarm>Outlier detected. Event IDs: 1 2 11</alarm>
<alarm>Outlier detected. Event IDs: 2 11 12</alarm>
<alarm>Outlier detected. Event IDs: 3 4 5</alarm>

```

XQuery #1

```

declare variable $seq external;
(: unordered = true; :)

for $x in distinct-values($seq/stream/event/@sensor)
  let $groupX := $seq/stream/event[@sensor eq $x]
return
  forseq $w in $groupX sliding window
    start position $s_pos when true()
    force end position $e_pos when $e_pos - $s_pos eq 2
  return
    let $maxTemp := max($w/@temp)
    let $minTemp := min($w/@temp)
    let $res := if ( $minTemp * 2 lt $maxTemp)
      then <alarm>Outlier detected. Event IDs:{data($w/@id)}</alarm>
      else ()
    return $res

```

Toolbox/outlier_detection2_1.xq

XQuery #2

```

declare variable $seq external;

forseq $w in $seq/stream/event sliding window
  start curItem $s_curr, position $s_pos when true()
  force end curItem $e_curr, position $e_pos when
    $s_curr/@sensor eq $e_curr/@sensor and
    count($seq[$s_pos to $e_pos][@sensor eq $e_curr/@sensor]) eq 3
return
  let $maxTemp := max($w/@temp)
  let $minTemp := min($w/@temp)
  let $res := if ( $minTemp * 2 lt $maxTemp)
    then <alarm>Outlier detected. Event IDs:
      {data($w[@sensor eq $e_curr/@sensor]/@id)}</alarm>
    else ()
  return $res

```

Toolbox/outlier_detection2_2.xq

XQuery #3

```

declare variable $seq external;

forseq $w in $seq/stream/event landmark window
  start curItem $s_curr when true()
  force end curItem $e_curr when $s_curr/@sensor eq $e_curr/@sensor
where count($w[@sensor eq $e_curr/@sensor]) eq 3
return

```

```

let $maxTemp := max($w/@temp)
let $minTemp := min($w/@temp)
let $res := if ( $minTemp * 2 lt $maxTemp)
  then <alarm>Outlier detected. Event IDs:
    {data($w[@sensor eq $e_curr/@sensor]/@id)}</alarm>
  else ( )
return $res

```

Toolbox/outlier_detection2_3.xq

Comments

Solution #1 requires that the query is set to “unordered” to allow an execution on an infinite stream.

6. RSS

RSS feeds are used to publish frequently updated digital content (e.g. blogs, news feeds or podcasts). In the following use cases we are going to present, how proposed XQuery extensions could be used to aggregate the feed content and to retrieve interesting information.

Note, that all use cases from section 3 Document Management (e.g. pagination) would be applicable to the feed aggregation results.

6.1 Annoying authors

Annoying authors.

The goal is to find all annoying authors who have posted three consecutive items in the RSS feed.

Input

```

<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Users\Tim\Desktop\rss-2_0.xsd">
  <channel>
    <title>DBIS RSS</title>
    <link>http://www.dbis.ethz.ch</link>
    <description>The forseq dummy RSS.</description>
    <language>en-us</language>
    <item>
      <title>Why use cases are important Part 1.</title>
      <category>Workshop</category>
      <author>rokas@e-mail.de</author>
      <pubDate>Tue, 03 Jun 2003 09:00:00 GMT</pubDate>
    </item>
    <item>
      <title>Why use cases are important Part 2.</title>
      <category>Workshop</category>
      <author>rokas@e-mail.de</author>
      <pubDate>Tue, 03 Jun 2003 09:00:00 GMT</pubDate>
    </item>
    <item>
      <title>Why use cases are important Part 3.</title>
      <category>Workshop</category>
      <author>rokas@e-mail.de</author>
      <pubDate>Tue, 03 Jun 2003 10:00:00 GMT</pubDate>
    </item>
    <item>
      <title>Extending XQuery with Window Functions</title>

```

| |
|---|
| <pre> <category>Talk</category> <author>tim@e-mail.de</author> <pubDate>Tue, 03 Jun 2003 11:00:00 GMT</pubDate> </item> <item> <title>XQueryP: A new programming language is born</title> <category>Talk</category> <author>david@e-mail.de</author> <pubDate>Tue, 03 Jun 2003 12:00:00 GMT</pubDate> </item> <item> <title>Why use cases are annoying to write.</title> <category>Talk</category> <author>rokas@e-mail.de</author> <pubDate>Wed, 04 Jun 2003 08:00:00 GMT</pubDate> </item> </channel> </rss> </pre> |
| Output |
| <pre><author>rokas@e-mail.de</author></pre> |
| XQuery |
| <pre> declare variable \$rssfeed external; forseq \$w in \$rssfeed/rss/channel/item tumbling window start curItem \$first when fn:true() end nextItem \$lookAhead when \$first/author ne \$lookAhead/author where count(\$w) ge 3 return \$w[1]/author RSS/annoying_authors.xq </pre> |
| Comments |
| |

6.2 Summary by category

| |
|--|
| Summary by category (periodical). |
| Every day provide summary of the RSS feed grouped by category. |
| Input |
| The same as in 6.1 Annoying authors example. |
| Output |
| <pre> <res> <date>2003-06-03</date> <category name="Workshop"><titles> <title>Why use cases are important Part 1.</title> <title>Why use cases are important Part 2.</title> <title>Why use cases are important Part 3.</title> </titles></category> <category name="Talk"><titles> <title>Extending XQuery with Window Functions</title> <title>XQueryP: A new programming language is born</title> </titles></category> </res> <res> <date>2003-06-04</date> <category name="Talk"><titles> <title>Why use cases are annoying to write.</title> </pre> |

| |
|--|
| <pre> </titles></category> </res> </pre> |
| <p>XQuery</p> <pre> declare variable \$rssfeed external; forseq \$w in \$rssfeed/rss/channel/item tumbling window start curItem \$s_curr when true() end nextItem \$e_next when fn:day-from-dateTime(xs:dateTime(\$e_next/pubDate)) ne fn:day-from-dateTime(xs:dateTime(\$s_curr/pubDate)) return <res> <date>{xs:date(xs:dateTime(\$s_curr/pubDate))}</date> { for \$c in fn:distinct-values(\$w/category) return <category name="{ \$c }"> <titles> { \$w[category eq \$c]/title } </titles> </category> } </res> </pre> <p>RSS/summary_by_category.xq</p> |
| <p>Comments</p> |

6.3 Summary by author

| |
|--|
| <p>Summary by author (periodical).</p> |
| <p>Every day provide summary of the RSS feed grouped by author.</p> |
| <p>Input</p> |
| <p>The same as in 6.1 Annoying authors example.</p> |
| <p>Output</p> |
| <pre> <res> <date>2003-06-03</date> <author name="rokas@e-mail.de"> <titles> <title>Why use cases are important Part 1.</title> <title>Why use cases are important Part 2.</title> <title>Why use cases are important Part 3.</title> </titles> </author> <author name="tim@e-mail.de"> <titles> <title>Extending XQuery with Window Functions</title> </titles> </author> <author name="david@e-mail.de"> <titles> <title>XQueryP: A new programming language is born</title> </titles> </author> </res> <res> </pre> |

```

<date>2003-06-04</date>
<author name="rokas@e-mail.de">
<titles>
  <title>Why use cases are annoying to write.</title>
</titles>
</author>
</res>

```

XQuery

```

declare variable $rssfeed external;

forseq $w in $rssfeed/rss/channel/item tumbling window
  start curItem $s_curr when true()
  end nextItem $e_next when
    fn:day-from-dateTime(xs:dateTime($e_next/pubDate)) ne
    fn:day-from-dateTime(xs:dateTime($s_curr/pubDate))
return
<res>
  <date>{xs:date(xs:dateTime($s_curr/pubDate))}</date>
  {
    for $a in fn:distinct-values($w/author)
    return
      <author name="{ $a }">
        <titles>
          { $w[author eq $a]/title }
        </titles>
      </author>
  }
</res>

```

RSS/summary_by_authors.xq

Comments

6.4 Interesting topics

Interesting topics.

Every day provide list of interesting topics in the RSS feed. In our example interesting means, that title of the item contains specific word (e.g. XQuery).

Input

The same as in 6.1 Annoying authors example.

Output

```

<res>
<date>2003-06-03</date>
  <title>Extending XQuery with Window Functions</title>
  <title>XQueryP: A new programming language is born</title>
</res>
<res><date>2003-06-04</date></res>

```

XQuery

```

declare variable $rssfeed external;

forseq $w in $rssfeed/rss/channel/item tumbling window
  start curItem $s_curr when true()
  end nextItem $e_next when
    fn:day-from-dateTime(xs:dateTime($e_next/pubDate)) ne
    fn:day-from-dateTime(xs:dateTime($s_curr/pubDate))
return

```

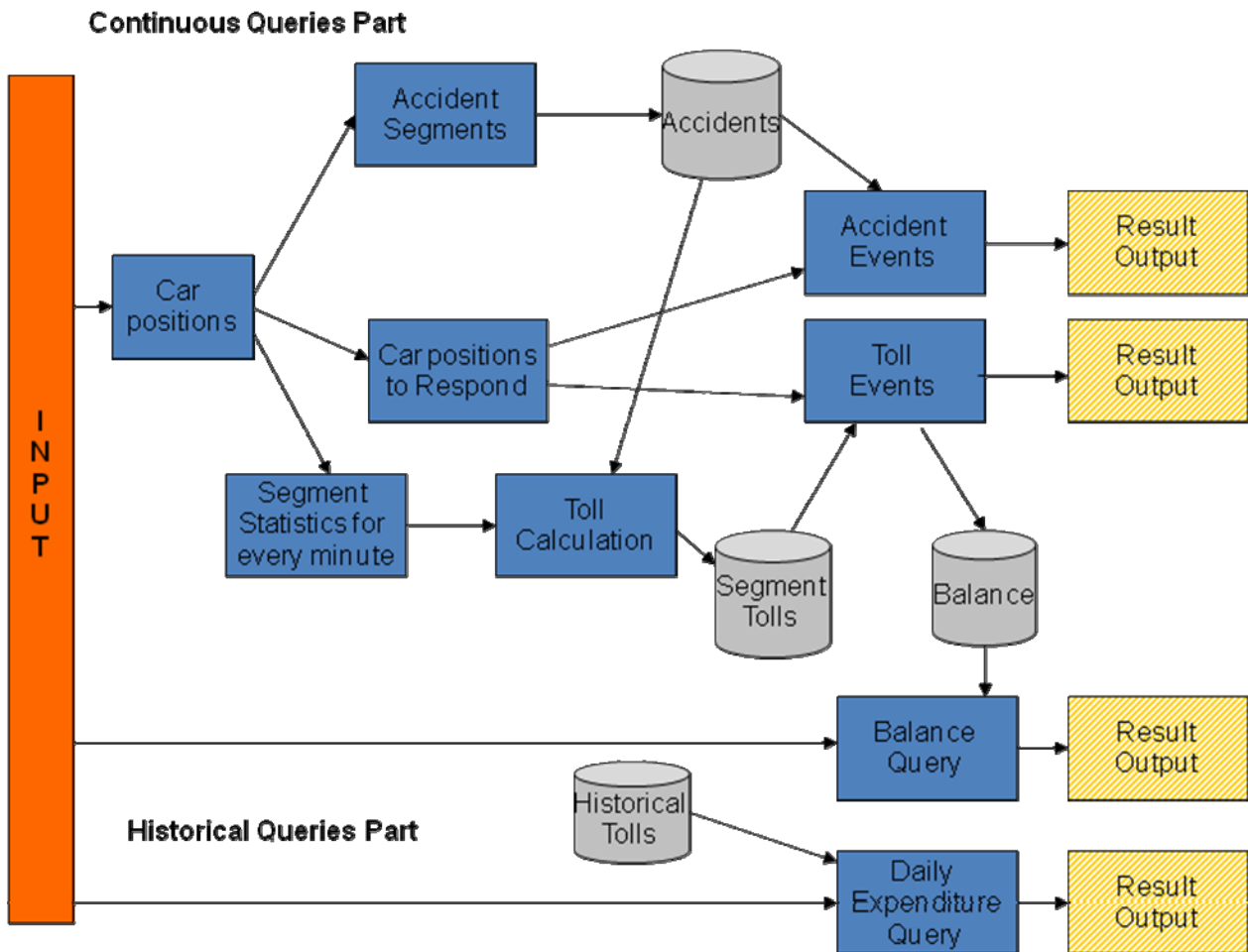
```
<res>
<date>{xs:date(xs:dateTime($s_curr/pubDate))}</date>
{
  for $item in $w
  where fn:contains( xs:string($item/title), 'XQuery')
  return $item/title
}
</res>
```

RSS/interesting_topics.xq

Comments

7. Linear Road Benchmark

The following use cases are used in our implementation of the Linear Road Benchmark [6]. Our first attempt was to implement the whole benchmark in a single XQuery expression; indeed, this is possible! However, our extended XQuery engine was not able to optimize this huge expression in order to achieve acceptable (i.e., compliant) performance. As a consequence, we decided to (manually) partition the implementation into nine continuous XQuery expressions and four (temporary) stores; i.e., a total of 13 “boxes”. The figure below shows the corresponding workflow.



The input stream produced by the Linear Road data generator feeds three continuous XQuery expressions which in turn generate streams that are fed into other XQuery expressions and intermediate stores. Binding an input stream to an XQuery expression is done by external variable declarations as specified in the XQuery recommendation. Seven threads are used in order to run the continuous XQuery expressions and move data into and out of data stores. Tightly coupled XQuery expressions (with a direct link in Figure) run in the same thread. As some of the queries use parts with side effects, we make use of the sequential mode introduced with XQueryP. As a matter of fact, it is also possible to implement LR without XQueryP, but our engine was not able to optimize those queries accordingly. We omit details of the way the synchronization between the boxes as this exceeds the scope of this document.

As input data, we used a CVS driven Iterator which produces one element with attributes out of one comma separate row. The following item sequence demonstrates how the data looks:

```
<rep Type="0 " Time="0 " VID="104 " Speed="20 " XWay="0 " Lane="0 " Dir="0
" Seg="3 " Pos="16178 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="101 " Speed="28 " XWay="0 " Lane="0 " Dir="0
" Seg="4 " Pos="21604 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="105 " Speed="26 " XWay="0 " Lane="0 " Dir="0
" Seg="9 " Pos="47954 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="110 " Speed="26 " XWay="0 " Lane="0 " Dir="0
" Seg="36 " Pos="190502 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="109 " Speed="29 " XWay="0 " Lane="0 " Dir="0
```

```

" Seg="37 " Pos="195836 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="114 " Speed="28 " XWay="0 " Lane="0 " Dir="0
" Seg="39 " Pos="206372 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="111 " Speed="23 " XWay="0 " Lane="0 " Dir="0
" Seg="40 " Pos="211574 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="102 " Speed="27 " XWay="0 " Lane="0 " Dir="0
" Seg="46 " Pos="243345 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="107 " Speed="26 " XWay="0 " Lane="0 " Dir="1
" Seg="73 " Pos="390292 " minute="0.5 " ></rep>
<rep Type="0 " Time="0 " VID="108 " Speed="23 " XWay="0 " Lane="0 " Dir="1
" Seg="68 " Pos="363940 " minute="0.5 " ></rep>

```

In the following, we present the queries of the different boxes:

| Carpositions |
|---|
| Filters car position events from all input events. |
| Query |
| <pre> declare variable \$InputSeq external; for \$w in \$InputSeq where \$w/@Type eq 0 return \$w </pre> |

| Segment Statistics for every Minute |
|--|
| |
| Description |
| Calculates segment statistics (average speed, number of cars) per minute which is used for the toll calculation. |
| Query |
| <pre> declare variable \$ReportedCarPositionsSeq external; forseq \$w in \$ReportedCarPositionsSeq early tumbling window start curItem \$s_curr, prevItem \$s_prev when (fn:ceiling(\$s_curr/@minute) ne fn:ceiling(\$s_prev/@minute)) end nextItem \$e_next when (\$s_curr/@minute +1) eq \$e_next/@minute let \$currMin := fn:ceiling(\$s_curr/@minute) let \$avgCarSpeed := for \$rep in \$w group \$rep as \$r-group by \$rep/@VID as \$vid_a, \$rep/@XWay as \$xway_a, \$rep/@Seg as \$seg_a, \$rep/@Dir as \$dir_a return <res XWay="{ \$xway_a }" Seg="{ \$seg_a }" Dir="{ \$dir_a }" VID="{ \$vid_a }" vAvgSpeed="{ avg(\$r-group/@Speed) }" ></res> let \$segStatistics := (<res endMark="1" minute="{ \$currMin }"></res>, for \$car in \$avgCarSpeed group \$car as \$c-group by \$car/@XWay as \$xway, \$car/@Seg as \$seg, \$car/@Dir as \$dir return <res endMark="0" minute="{ \$currMin }" XWay="{ \$xway }" Seg="{ \$seg }" Dir="{ \$dir }" avgSpeed="{ avg(\$c-group/@vAvgSpeed) }" carCount="{ count(\$c- group) }"></res>, <res endMark="3" minute="{ \$currMin }"></res>) return \$segStatistics </pre> |

| Comments |
|----------|
| |

| Car position to respond (v.1) |
|--|
| Car position reports which require system response (car crossed the segment). |
| Query |
| <pre> declare variable \$ReportedCarPositionsSeq external; forseq \$w in \$ReportedCarPositionsSeq sliding window start curItem \$s_curr, position \$s_p when true end curItem \$e_curr, position \$e_p when (\$s_curr/@VID eq \$e_curr/@VID and \$e_p ne \$s_p) or (\$e_curr/@Time gt \$s_curr/@Time + 30) where (\$s_curr/@VID eq \$e_curr/@VID) and (\$s_curr/@Seg ne \$e_curr/@Seg and data(\$e_curr/@Lane) ne 4) return \$e_curr </pre> |
| Comments |
| |

| Car position to respond (v.2) |
|---|
| To speed up the previous <i>Car position to respond</i> query there is an alternative solution implemented with XQueryP (it uses an additional store). |
| Query |
| <pre> declare execution sequential; declare variable \$ReportedCarPositionsSeq external; declare variable \$CAR_POS_STORAGE external; for \$item in \$ReportedCarPositionsSeq let \$prevCarRep := \$CAR_POS_STORAGE[@VID eq \$item/@VID] return (if (count(\$prevCarRep) eq 0 or (\$prevCarRep/@Seg ne \$item/@Seg and \$item/@Lane ne 4)) then \$item else (), replace node \$CAR_POS_STORAGE[@VID eq \$item/@VID] with \$item) </pre> |
| Comments |
| |

| Accident Segments |
|--|
| This query detects accidents in the particular segments of the expressway. |
| Query |
| <pre> declare execution sequential; declare variable \$ReportedCarPositionsSeq external; forseq \$w in \$ReportedCarPositionsSeq early sliding window start curItem \$s_curr, prevItem \$s_prev when (\$s_curr/@minute) ne (\$s_prev/@minute) end nextItem \$e_next when (\$s_curr/@minute +2) eq (\$e_next/@minute) let \$currMin := (\$s_curr/@minute) +1 let \$stopedCars := for \$rep in \$w group \$rep as \$r-group by \$rep/@VID as \$vid_s, \$rep/@XWay as \$xway_s, </pre> |

```

$rep/@Seg as $seg_s, $rep/@Dir as $dir_s, $rep/@Lane as $lane_s, $rep/@Pos as
$pos_s
  where count($r-group) ge 4
  return <stopped_car VID="{ $vid_s}" XWay="{ $xway_s}" Seg="{ $seg_s}"
Dir="{ $dir_s}" Lane="{ $lane_s}" Pos="{ $pos_s}"></stopped_car>

let $accidents :=
  for $car in $stoppedCars
  group $car as $c-group by $car/@XWay as $xway_a, $car/@Seg as $seg_a,
$car/@Dir as $dir_a, $car/@Lane as $lane_a, $car/@Pos as $pos_a
  where count($c-group) ge 2
  return <accident minute="{ $currMin}" XWay="{ $xway_a}" Seg="{ $seg_a}"
Dir="{ $dir_a}"></accident>
let $accidentsRes := if ( count($accidents) gt 0 ) then <accidents>
{$accidents} </accidents>
  else <accidents><accident minute="{ $currMin}" XWay="-1" Seg="-1" Dir="-
1"></accident></accidents>

return $accidentsRes
do delete $ACCIDENT_STORAGE[@minute < $currMin];
do insert $accidentsRes into $ACCIDENT_STORAGE

```

Comments

Toll Calculation

Based on the accident and the segment statistics information the toll is calculated.

Query

```

declare execution sequential;
declare variable $SegmentStatSeq external;
declare variable $ACCIDENT_STORAGE external;
declare variable $TOLL_STORAGE external;

forseq $w in $SegmentStatSeq sliding window
  start prevItem $s_prev when $s_prev/@endMark eq 1
  force end nextItem $e_next when ($e_next/@endMark eq 3) and (
($s_prev/@minute + 4) eq $e_next/@minute )

  let $resMin := $e_next/@minute
  let $allAccSeg := $ACCIDENT_STORAGE[@minute eq $resMin]

  let $segData :=
    for $s in $w
    where $s/@endMark eq 0
    group $s as $s-group by $s/@XWay as $xway, $s/@Seg as $seg, $s/@Dir as
$dir
    return
      <res XWay="{ $xway}" Seg="{ $seg}" Dir="{ $dir}" avgSpeed="{ avg($s-
group/@avgSpeed) }"
      carCount="{ $s-group[@minute eq $resMin]/@carCount}"></res>

  let $allAffectedSeg :=
  for $segmCurr in $allAccSeg
  let $segm := $segmCurr/@Seg
  return
    if ($segmCurr/@Dir eq 0)
    (: eastbound direction :)
    then <res>

```

```

<XWay>{data($segmCurr/@XWay)}</XWay><Dir>{data($segmCurr/@Dir)}</Dir><startSeg>
{data($segm) - 4}</startSeg><endSeg>{data($segm)}</endSeg> </res>
  (: westbound direction :)
  else <res>
<XWay>{data($segmCurr/@XWay)}</XWay><Dir>{data($segmCurr/@Dir)}</Dir><startSeg>
{data($segm)}</startSeg><endSeg>{data($segm) + 4}</endSeg> </res>

let $tollResults :=
  for $sData in $segData

    let $affSeg :=
      for $sCurr in $allAffectedSeg
        where $sCurr/XWay eq $sData/@XWay and $sCurr/Dir eq $sData/@Dir and
        $sCurr/startSeg le $sData/@Seg and $sCurr/endSeg ge $sData/@Seg
      return $sCurr
    let $notInAccidentZone := count($affSeg) eq 0

    let $lastMinCarCount := $sData/@carCount - 50
    let $t := if ( $notInAccidentZone and $sData/@avgSpeed < 40 and
    $lastMinCarCount > 0 )
      then $lastMinCarCount * $lastMinCarCount * 2
      else 0
    return <res minute="{ $resMin + 1}" XWay="{data($sData/@XWay)}"
    Seg="{data($sData/@Seg)}" Dir="{data($sData/@Dir)}"
    avgSpeed="{data($sData/@avgSpeed)}" ccount="{data($sData/@carCount)}"
    toll="{ $t}"> </res>

return
  do delete $TOLL_STORAGE[@minute < $resMin];
  do insert $tollResults into $TOLL_STORAGE

```

Comments

Accident Events

The cars have to be notified if they are getting close to the accident zone.

Query

```

declare variable $ACCIDENT_STORAGE external;
declare variable $CAR_POSITIONS_TO_RESPOND external;

for $s_curr in $CAR_POSITIONS_TO_RESPOND

  let $prevMin := $s_curr/@Time idiv 60

  let $allAccSegOnWay := $ACCIDENT_STORAGE[@XWay eq $s_curr/@XWay and @Dir eq
  $s_curr/@Dir and @minute eq $prevMin]
  let $checkAcc :=
    for $s in $allAccSegOnWay/@Seg
      let $accOnWay := if ($s_curr/@Dir eq 0)
        (: eastbound direction :)
        then if ( ($s -5) lt $s_curr/@Seg and $s_curr/@Seg le $s) then
data($s) else()
        (: westbound direction :)
        else if ($s +5 gt $s_curr/@Seg and $s_curr/@Seg ge $s) then data($s)
      else()
    return $accOnWay

  let $accidentAlert := if ( count($checkAcc) gt 0 )
    then <alert Type="1" Time="{ $s_curr/@Time}" Emit="" VID="{ $s_curr/@VID}"

```

| |
|--|
| <pre> Seg="{ \$checkAcc[1] }"></alert> else () return \$accidentAlert </pre> |
| Comments |
| |

| |
|---|
| Toll Events |
| <p>This query notifies the car about the toll to be charged, when a car is entering into the new segment. It also calculates the current toll balance of the car (to the current balance it adds the toll charged for the previous segment). This query could be written without the sequential mode but then it would be extremely hard to optimize.</p> |
| Query |
| <pre> declare execution sequential; declare variable \$TOLL_STORAGE external; declare variable \$BALANCE_STORAGE external; declare variable \$CAR_POSITIONS_TO_RESPOND external; for \$s_curr in \$CAR_POSITIONS_TO_RESPOND let \$prevMin := (\$s_curr/@Time idiv 60) + 1 let \$segToll := \$TOLL_STORAGE[@Seg eq \$s_curr/@Seg and @XWay eq \$s_curr/@XWay and @Dir eq \$s_curr/@Dir and @minute eq \$prevMin] let \$toll := if (count(\$segToll) eq 0) then 0 else data(\$segToll/@toll) let \$speed := if (count(\$segToll) eq 0) then 0 else data(\$segToll/@avgSpeed) let \$oldValue := \$BALANCE_STORAGE[@VID eq \$s_curr/@VID] let \$newBalance := \$oldValue/@toll + \$oldValue/@balance let \$newValue := <res VID="{ \$s_curr/@VID }" Time="{ \$s_curr/@Time }" Bal="{ \$newBalance }" Toll="{ \$toll }"></res> do replace \$BALANCE_STORAGE[@VID eq \$s_curr/@VID] with \$newValue return <res Type="0" VID="{ \$s_curr/@VID }" Time="{ \$s_curr/@Time }" Emit="" Speed="{ \$speed }" Toll="{ \$toll }"></res> </pre> |
| Comments |
| |

| |
|--|
| Balance Query |
| <p>This query returns the current toll balance for an incoming balance request.</p> |
| Query |
| <pre> declare variable \$InputSeq external; declare variable \$BALANCE_STORAGE external; for \$w in \$InputSeq where \$w/@Type eq 2 return let \$carBal := \$BALANCE_STORAGE[@VID eq \$w/@VID] return <res Type="2" Time="{ \$w/@Time }" Emit="" ResultTime="{ \$carBal/@Time }" QID="{ \$w/@Qid }" Bal="{ \$carBal/@Bal }"></res> </pre> |
| Comments |
| |

8. Other scenarios

Possible further use cases:

- Queries on astronomical data (compare e.g. running example of Li: Efficient Evaluation of XQuery over Streaming Data) or biological pathway data (BioPAX)
- Telecommunication management
- Network monitoring
- Weblog: For each user maintain the number of actions per session. (Session defined by a login + logout.)
- Smart Home moods (status, control and orchestration of the devices), house owner notifications when subscribed event happens, baby monitoring.
- Complex Event Processing
- Business Activity Monitoring (BAM)
- provide statistics on business process performance.
- provide instant insight into how IT events at any system level (e.g., network failures, database access loads, on-line website activity, and changes in metrics on all kinds of resources) will affect the progress of high level business transactions.
- automate real-time notification of violation or pending violation of business level policies (e.g., SLAs).
- A commercial broker system Brokers are software systems that mediate between entities such as service requestors and service providers. They can be viewed as an example of applying virtual XML to the data broker pattern. Messaging brokers are used to implement the event-driven and XML-based messaging engine (the bus) of the Enterprise Service Bus

And more further ideas:

- <http://voipforenterprise.tmcnet.com/feature/Enterprise-transformation/articles/3179-implementing-complex-event-processing-two-use-cases.htm>

- <http://complexevents.com/?p=20> : “Complex” or “Simple” Event Processing
- <http://complexevents.com/?p=24> : A Challenge for the BAM Industry

9. Bibliography

- [1] I. Botan, Peter M. Fischer, D. Florescu, D. Kossmann, T. Kraska and R. Tamosevicius. *Extending XQuery with Window Functions*. VLDB 2007, to appear.
Available at: <http://www.dbis.ethz.ch/research/publications>
- [2] V. Borkar and M. Carey. *Extending XQuery for Grouping, Duplicate Elimination, and Outer Joins*. URL: <http://www.idealliance.org/proceedings/xml04/abstracts/paper229.html> Last visited: 2007-09-18.
- [3] Don Chamberlin, Daniela Florescu, Jim Melton, Jonathan Robie, and Jérôme Siméon: *XQuery Update Facility 1.0* URL: <http://www.w3.org/TR/xquery-update-10/>. Last visited: 2007-09-18.
- [4] D. Chamberlin, M. Carey, D. Florescu, D. Kossmann and J. Robie. XQueryP: Programming with XQuery. In *Proceedings of the Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)* Jun 30, Chicago, USA, ACM, 2006.
- [5] M. Kay. Positional Grouping in XQuery. In *Proceedings of the Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)* Jun 30, Chicago, USA, ACM, 2006.
- [6] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker and R. Tibbetts. Linear Road: A Stream Data Management Benchmark. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, Aug 31 - Sep 3, Toronto, Canada, pp 480-491, Morgan Kaufmann, 2004.