The CQL continuous query language: semantic foundations and query execution

A. Arasu, S. Babu, J. Widom

VLDBJ 06

Presented by Liping Peng
## Databases vs Data streams

- Query-driven processing v.s. Data-driven processing

### DBMS
- **Model:** persistent data
- **Table:** set/bag of tuples
- **Updates:** all
- **Query:** transient
- **Query Answer:** exact
- **Query Eval:** multi-pass
- **Operator:** blocking OK
- **Query Plan:** fixed

### DSMS
- **Model:** transient data
- **Infinite sequence** of tuples
- **Updates:** append only
- **Query:** persistent
- **Query Answer:** often approx [?]
- **Query Eval:** one-pass
- **Operators:** unblocking only
- **Query Plan:** adaptive
Two Data Types

- **Schema**: a fixed set of named attributes
- **Time domain**: discrete, ordered

- **Stream**: a (finite/infinite) bag of \(<s, \tau>\)
- **Relation**: an unordered bag of tuples at each time instant in time domain
Three Classes of Operators

- **stream-to-relation**: one stream -> one relation
- **relation-to-relation**: one or more relations -> one relation
- **relation-to-stream**: one relation -> one stream
Stream-to-Relation operators

(1) time-based sliding window

- **Syntax:** \( S [\text{Range } T] \)

- **Semantics:** for each \( \tau \), time-based sliding window keeps tuples that arrive no earlier than \( T \) time units before \( \tau \).

\[
R(\tau) = \{ s \mid (s, \tau') \in S \land (\tau' \leq \tau) \land (\tau' \geq \max\{\tau - T, 0\}) \}
\]

- **Special cases:**
  - \( S [\text{Now}] \) (\( T=0 \)): tuples with timestamp \( \tau \)
  - \( S [\text{Range Unbounded}] \) (\( T=\infty \)): all tuples up to \( \tau \)
Stream-to-Relation operators

(1) time-based sliding window

\begin{align*}
\text{tuple, time} & : \langle (a_0), 0 \rangle, \langle (a_1), 1 \rangle, \langle (a_2), 2 \rangle, \langle (a_3), 2 \rangle, \langle (a_4), 3 \rangle, \langle (a_5), 5 \rangle \\
S \text{ [Now]} & : (a_0), (a_1), (a_2), (a_3), (a_4), (a_5) \\
S \text{ [Range 2]} & : (a_0), (a_1), (a_2) \\
S \text{ [Range Unbounded]} & : (a_0), (a_1), (a_2) 
\end{align*}
Stream-to-Relation operators

(2) tuple-based sliding window

- Syntax: $S \left[ \text{Rows } N \right]$

- Semantics: for each $\tau$
  
  $R(\tau) = \{ \text{the last } N \text{ tuples of } S \text{ that have been seen at time } \tau \}$

- Special cases:
  - $S \left[ \text{Rows Unbounded} \right] \ (N=\infty) = S \left[ \text{Range Unbounded} \right]$
    
    all tuples that have been seen
Stream-to-Relation operators

(2) tuple-based sliding window

\[ \text{t = 0} \rightarrow \{ (a_0), 0 \} \]
\[ \text{t = 1} \rightarrow \{ (a_0), 1 \} \]
\[ \text{t = 2} \rightarrow \{ (a_2), 2 \} \]
\[ \text{t = 3} \rightarrow \{ (a_3), 2 \} \]
\[ \text{t = 4} \rightarrow \{ (a_4), 3 \} \]
\[ \text{t = 5} \rightarrow \{ (a_5), 5 \} \]
Stream-to-Relation operators
(3) partitioned sliding window

- Syntax: $S \text{ [Partition By } A_1, \ldots, A_k \text{ Rows N]}$

- Semantics: group by $A_1, \ldots, A_k$, keep the latest $N$ tuples for each group and merge them together
Stream-to-Relation operators (3) partitioned sliding window

<table>
<thead>
<tr>
<th>t=0</th>
<th>(vid,spd),time</th>
<th>vid = 1</th>
<th>vid = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>S [Partition By vid Rows 2]</td>
<td>(1,a₀), 0</td>
<td>(1,a₀)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1,a₁), 1</td>
<td>(1,a₁)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(1,a₂), 2</td>
<td>(1,a₂)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2,a₃), 2</td>
<td></td>
<td>(2,a₃)</td>
</tr>
<tr>
<td></td>
<td>(1,a₄), 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2,a₅), 5</td>
<td></td>
<td>(2,a₅)</td>
</tr>
</tbody>
</table>
Stream-to-Relation operators

optional “slide” parameter

- Use “slide” parameter to tune the granularity
- Example: S [Rows 3, slide 2]
Relation-to-Stream operators

- **Istream** [insert stream]
  \[ Istream(R) = \bigcup_{\tau \geq 0} ((R(\tau) - R(\tau - 1)) \times \{\tau\}) \]

- **Dstream** [delete stream]
  \[ Dstream(R) = \bigcup_{\tau > 0} ((R(\tau - 1) - R(\tau)) \times \{\tau\}) \]

- **Rstream** [relation stream]
  \[ Rstream(R) = \bigcup_{\tau \geq 0} (R(\tau) \times \{\tau\}) \]

- **Cartesian Product**
$$\text{Istream}(R) = \bigcup_{\tau \geq 0} ((R(\tau) - R(\tau - 1)) \times \{\tau\})$$

Select Istream(*)
From PosSpeedStr [Range Unbounded]
Where speed > 65

Output: $\langle(1,70),0\rangle$ $\langle(1,70),1\rangle$ $\langle(2,70),2\rangle$ $\langle(1,80),3\rangle$
\[ R_{\text{stream}}(R) = \bigcup_{\tau \geq 0} (R(\tau) \times \{\tau\}) \]

Select \( R_{\text{stream}}(\ast) \)
From PosSpeedStr [Now]
Where speed > 65

\[ t = \emptyset \]

Output:
\[(1,70), 0> \quad (1,70), 1> \quad (1,50), 2> \quad (2,70), 2> \quad (1,80), 3> \quad (2,50)]
\[ Dstream(R) = \bigcup_{\tau > 0} (R(\tau - 1) - R(\tau)) \times \{\tau\} \]

Select Dstream(*)
From PosSpeedStr [Range 2]

\[ t=0 \quad t=1 \quad t=2 \quad t=3 \quad t=4 \quad t=5 \]

output: \((1,70),3\) \((1,70),4\) \((2,70),5\)
Select Dstream(vid) 
From PosSpeedStr [Range 2]

\[
Dstream(R) = \bigcup_{\tau > 0} ((R(\tau - 1) - R(\tau)) \times \{\tau\})
\]

Output: \((1, 4)\) \((2, 5)\)
CQL Implementation:
Representation of streams and relations

- Append-only tagged-tuple sequence
- Example:

<table>
<thead>
<tr>
<th>S</th>
<th>S [Range 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;a_0&gt;, 0&gt;)</td>
<td>(&lt;a_0&gt;, 0, true&gt;)</td>
</tr>
<tr>
<td>(&lt;a_1&gt;, 1&gt;)</td>
<td>(&lt;a_1&gt;, 1, true&gt;)</td>
</tr>
<tr>
<td>(&lt;a_2&gt;, 2&gt;)</td>
<td>(&lt;a_2&gt;, 2, true&gt;)</td>
</tr>
<tr>
<td>(&lt;a_3&gt;, 2&gt;)</td>
<td>(&lt;a_3&gt;, 2, true&gt;)</td>
</tr>
<tr>
<td>(&lt;a_4&gt;, 3&gt;)</td>
<td>(&lt;a_4&gt;, 3, true&gt;)</td>
</tr>
<tr>
<td></td>
<td>(&lt;a_0&gt;)</td>
</tr>
<tr>
<td></td>
<td>(&lt;a_1&gt;)</td>
</tr>
<tr>
<td></td>
<td>(&lt;a_2&gt;)</td>
</tr>
<tr>
<td></td>
<td>(&lt;a_3&gt;)</td>
</tr>
<tr>
<td></td>
<td>(&lt;a_4&gt;)</td>
</tr>
</tbody>
</table>
CQL Implementation: Operators

- **Data Operators**
  - stream-to-relation: seq-window (time-based, tuple-based, partitioned)
  - relation-to-relation: select, project, join, aggregate, union, intersect, .etc.
  - relation-to-stream: Istream, Dstream, Rstream

- **System Operators**
  - Isolate data operators from various low-level issues
  - Examples:
    - stream-shepherd: handles input streams arriving over the network
    - output: send results to remote clients
CQL Implementation: Operators (cont.)

- Each operator has
  - input queue(s) [the output queue(s) of the preceding operator(s)]
  - output queue(s) [the input queue(s) of the subsequent operator(s)]
  - [one or more synopses]
    - a hash-table for each input of an equi-join
    - tuple buffer: tuples in the current window
Q1: Select B, max(A)  
From S1 [Rows 50,000]  
Group By B  

Q2: Select Istream(*)  
From S1 [Rows 40,000], S2 [Range 600 Seconds]  
Where S1.A = S2.A

Data comes from the bottom and is pushed to be processed in a bottom-up way.

Queues between two consecutive operators.

Shared downstream operators and synopses.
Q & A