The CQL Continuous Query Language: Semantic Foundations and Query Execution

Arvind Arasu, Shivnath Babu, Jennifer Widom

Presented by: Young-Rae Kim
INTRODUCTION
CQL – Continuous Query Language
“Many papers include example continuous queries expressed in some declarative language .... However, ... a precise language semantics, ... often is left unclear.”[1]
2003: A CQL Odyssey

“For furthermore, very little has been published to date covering execution details of general-purpose continuous queries.” [1]
“It may appear initially that defining a continuous query language over (relational) streams is not difficult .... However, as queries get more complex ... the situation becomes much murkier.” [1]
It’s all about the abstract semantics

CQL abstract semantics is based on

2 data types
• Streams
• Relations

3 classes of operators
• Stream-to-Relation
• Relation-to-Relation
• Relation-to-Stream
It’s all about the abstract semantics – Goals

1. Make it easy to understand; familiar.

2. Simple queries should be easy to write, compact, and shouldn’t be visually deceiving.
Query Execution Plans Matter Too! – Goals

1. Plans should consist of modular and pluggable components based on generic interfaces

2. An execution model that efficiently captures the combination of streams and relations

3. An architecture that makes performance-based experimentation easy.
DEFINITIONS

The technical stuff.
Streams vs. Relations

Streams

• “a (possibly infinite) bag (multiset*) of elements <s, τ>”[2]

Relations

• “A relation R is a mapping from T to a finite but unbounded bag of tuples belonging to the schema of R.”[3]
Abstract Semantics

Continuous Semantics

• Assume a discrete, ordered time domain T.
• Inputs are either streams or relations
• In discussing the result of a continuous query $Q$ at a time $\tau$, there are 2 possibilities:
  1. The outermost operator in $Q$ is relation-to-stream. The result of $Q$ at time $\tau$ is $S$ (the produced stream) up to $\tau$.
  2. The outermost operator in $Q$ is stream-to-relation or relation-to-relation. The result of $Q$ at time $\tau$ is $R(\tau)$ (the produced relation).
• Time only “advances” from $\tau$ from $(\tau - 1)$ when all inputs up to $\tau - 1$ have been processed.
What we’re all here for.

CQL
Operators: Stream-to-Relation

- Based on the concept of a *sliding window* over a stream.
- SQL-99 derivative.
- 3 classes:
  1. Time-based
     "S [Range T]"
  2. Tuple-based
     "S [Rows N]"
  3. Partitioned
     "S [Partition By A1, \ldots, Ak rows N]"
Operators:
Relation-to-Relation

• Derived from traditional SQL.
Operators: Relation-to-Stream

• 3 operators:
  - Istream ("insert stream")
    \[
    \text{Istream}(R) = \bigcup_{\tau \geq 0} ((R(\tau) - R(\tau - 1)) \times \{\tau\})
    \]
  - Dstream ("delete stream")
    \[
    \text{Dstream}(R) = \bigcup_{\tau > 0} ((R(\tau - 1) - R(\tau)) \times \{\tau\})
    \]
  - Rstream ("relation stream")
    \[
    \text{Rstream}(R) = \bigcup_{\tau \geq 0} (R(\tau) \times \{\tau\})
    \]
Operators:
Example

```
SELECT Istream(*)
FROM PosSpeedStr [Range Unbounded]
WHERE speed > 65
```
Shortcuts & Defaults

Default Windows

• **Unbounded** windows are applied to streams by default.

Default Relation-to-Stream Operators

• An intended **Istream** operator may be omitted from a CQL query.
Post Shortcut Query Example

SELECT *
FROM PosSpeedStr
WHERE speed > 65
Equivalences

• Important for query-rewrite optimizations
• All equivalences that hold in SQL with standard relational semantics carry over to the relational portion of CQL.
• 2 stream-based equivalences in CQL:
  1. Window reduction
  2. Filter-window commutativity
Equivalences: Window Reduction

SELECT Istream(L)
FROM S [Range Unbounded]
WHERE C

is equivalent to

SELECT Rstream(L)
FROM S [Now]
WHERE C
Equivalences: Window Reduction

SELECT Istream(L)
FROM S [Range Unbounded]
WHERE C

is equivalent to

SELECT Rstream(L)
FROM S [Now]
WHERE C
Equivalences:
Filter-Window Commutativity

(SELECT L
FROM S
WHERE C) [Range T]

is equivalent to

SELECT L
FROM S [Range T]
WHERE C
Equivalences:

Filter-Window Commutativity

\[(\text{SELECT } L \\text{ FROM } S \\text{ WHERE } C) \ [\text{Range } T]\]

is equivalent to

\[\text{SELECT } L \\text{ FROM } S \ [\text{Range } T] \\text{ WHERE } C\]
Equivalences:
Filter-Window Commutativity

(SELECT L
  FROM S
  WHERE C) [Range T]

is equivalent to

SELECT L
  FROM S [Range T]
  WHERE C
Time Management

• More realistic conditions: we make the aforementioned assumption.
  – The network conveying the stream elements to the DSMS may not guarantee in-order transmission
  – Streams pause and restart

• Use additional “meta-input” to the system to cope
  – ‘heartbeats’ in STREAM
Time Management: Heartbeats

- A heartbeat consists simply of a timestamp \( \tau \in T \).
- After arrival of the heartbeat, the system will reject stream elements with timestamp \( \leq \tau \).
- Various ways to generate heartbeats
Environment Overview
CQL IMPLEMENTATION IN STREAM

Hot and STREAM-y.
STREAM Query Plans

• Each query plan runs continuously and is composed of 3 different types of components:
  – Operators
  – Queues
  – Synopses
Operators

• Read from one or more input queues, processes the input based on its semantics, and writes its output to an output queue.

• In STREAM, every operator is either a CQL operator or a system operator.
# Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-window</td>
<td>stream-to-relation</td>
<td>Implements time-based, tuple-based, and partitioned windows</td>
</tr>
<tr>
<td>select</td>
<td>relation-to-relation</td>
<td>Filters tuples based on predicate(s)</td>
</tr>
<tr>
<td>project</td>
<td>relation-to-relation</td>
<td>Duplicate-preserving projection</td>
</tr>
<tr>
<td>binary-join</td>
<td>relation-to-relation</td>
<td>Joins two input relations</td>
</tr>
<tr>
<td>mjoin</td>
<td>relation-to-relation</td>
<td>Multiway join from [VNB03]</td>
</tr>
<tr>
<td>union</td>
<td>relation-to-relation</td>
<td>Bag union</td>
</tr>
<tr>
<td>except</td>
<td>relation-to-relation</td>
<td>Bag difference</td>
</tr>
<tr>
<td>intersect</td>
<td>relation-to-relation</td>
<td>Bag intersection</td>
</tr>
<tr>
<td>antisemijoin</td>
<td>relation-to-relation</td>
<td>Antisemijoin of two input relations</td>
</tr>
<tr>
<td>aggregate</td>
<td>relation-to-relation</td>
<td>Performs grouping and aggregation</td>
</tr>
<tr>
<td>duplicate-eliminate</td>
<td>relation-to-relation</td>
<td>Performs duplicate elimination</td>
</tr>
<tr>
<td>i-stream</td>
<td>relation-to-stream</td>
<td>Implements Istream semantics</td>
</tr>
<tr>
<td>d-stream</td>
<td>relation-to-stream</td>
<td>Implements Dstream semantics</td>
</tr>
<tr>
<td>r-stream</td>
<td>relation-to-stream</td>
<td>Implements Rstream semantics</td>
</tr>
<tr>
<td>stream-shepherd</td>
<td>system operator</td>
<td>Handles input streams arriving over the network</td>
</tr>
<tr>
<td>stream-sample</td>
<td>system operator</td>
<td>Samples specified fraction of tuples</td>
</tr>
<tr>
<td>stream-glue</td>
<td>system operator</td>
<td>Adapter for merging a stream-producing view into a plan</td>
</tr>
<tr>
<td>rel-glue</td>
<td>system operator</td>
<td>Adapter for merging a relation-producing view into a plan</td>
</tr>
<tr>
<td>shared-rel-op</td>
<td>system operator</td>
<td>Materializes a relation for sharing</td>
</tr>
<tr>
<td>output</td>
<td>system operator</td>
<td>Sends results to remote clients</td>
</tr>
</tbody>
</table>
Queues

- Connect its input operator to its output operator.
- Stored entirely in memory.*
Synopses

• Store the intermediate state needed by continuous query plans.
  – E.g. performing a windowed join of two streams

• Many synopses are logical “stubs” that primarily point into other synopses.

• Most common use of a synopsis is to materialize the current state of a relation.

• Also stored entirely in memory.*
Example Query Plan

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
Query Optimization

• Naïve query plan generator.

• Commonly applied heuristics:
  – Push selections below joins
  – Maintain and use indexes for synopses on binary-join, mjoin, and aggregate operators.
  – Share synopses within query plans whenever possible.
We’re #1.

COMPARISON WITH OTHER LANGUAGES
Tapestry

- Expressed using SQL syntax.
- Does not support sliding windows over streams or any relation-to-stream operators.
Tribeca

• Based on stream-to-stream operators.
• Queries take a single stream as input and produce a single stream as output, with no notion of relation.
Aurora

• Difficult to compare the procedural query interface of Aurora against a declarative language (CQL).

• Some distinctions:
  – Aggregation operators defined by user-defined functions and have optional parameters set by the user
  – Aurora does not explicitly support relations.
TelegraphCQ (Stream-Only)

• Note that we can derive a stream-only language in CQL anyways.

• Motivations for CQL’s dual approach:
  – Make it easy to understand; familiar.
  – More intuitive queries.
  – Use of both relations and streams cleanly generalizes materialized views.
El finito.

THE END
Image Sources (in order)

- http://www.harborfreight.com/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/im/image_16157.jpg
- http://bethesdalutherancommunities.org/view.image?Id=2216
- http://wiki.treck.com/images/d/d4/Fig1.39_Sliding_Window_Protocol.gif
- U. Srivastava and J. Widom. Flexible time management in data stream systems. Page 264
- https://edc2.healthtap.com/ht-staging/user_answer/avatars/211738/large/What_is_a_Steam_Shower_8666546_460.jpeg?1386553186
- The CQL Continuous Query Language: Semantic Foundations and Query Execution, Page 25
- The CQL Continuous Query Language: Semantic Foundations and Query Execution, Page 22
- http://28.media.tumblr.com/tumblr_le900rkVhp1qzexono1_500.png
- https://40.media.tumblr.com/3290de4ee413685c60f08dc775310524/tumblr_mydnbyvVBr1rz41foxo2_400.jpg