The CQL Continuous Query Language: Semantic Foundations and Query Execution

Arvind Arasu,
Shivnath Babu,
Jennifer Widom

Presented by: Young-Rae Kim

You are here!

INTRODUCTION

CQL – **C**ontinuous **Q**uery **L**anguage

2003: A CQL Odyssey

“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

"Furthermore, very little has been published to date covering execution details of general-purpose continuous queries." [1]

2003: A CQL Odyssey

"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

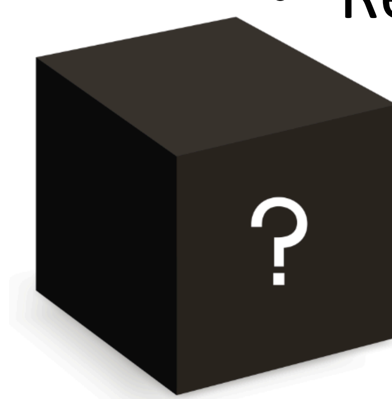
CQL abstract semantics is based on

<u>2 data types</u>

- Streams
- Relations

<u>3 classes of operators</u>

- 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.*

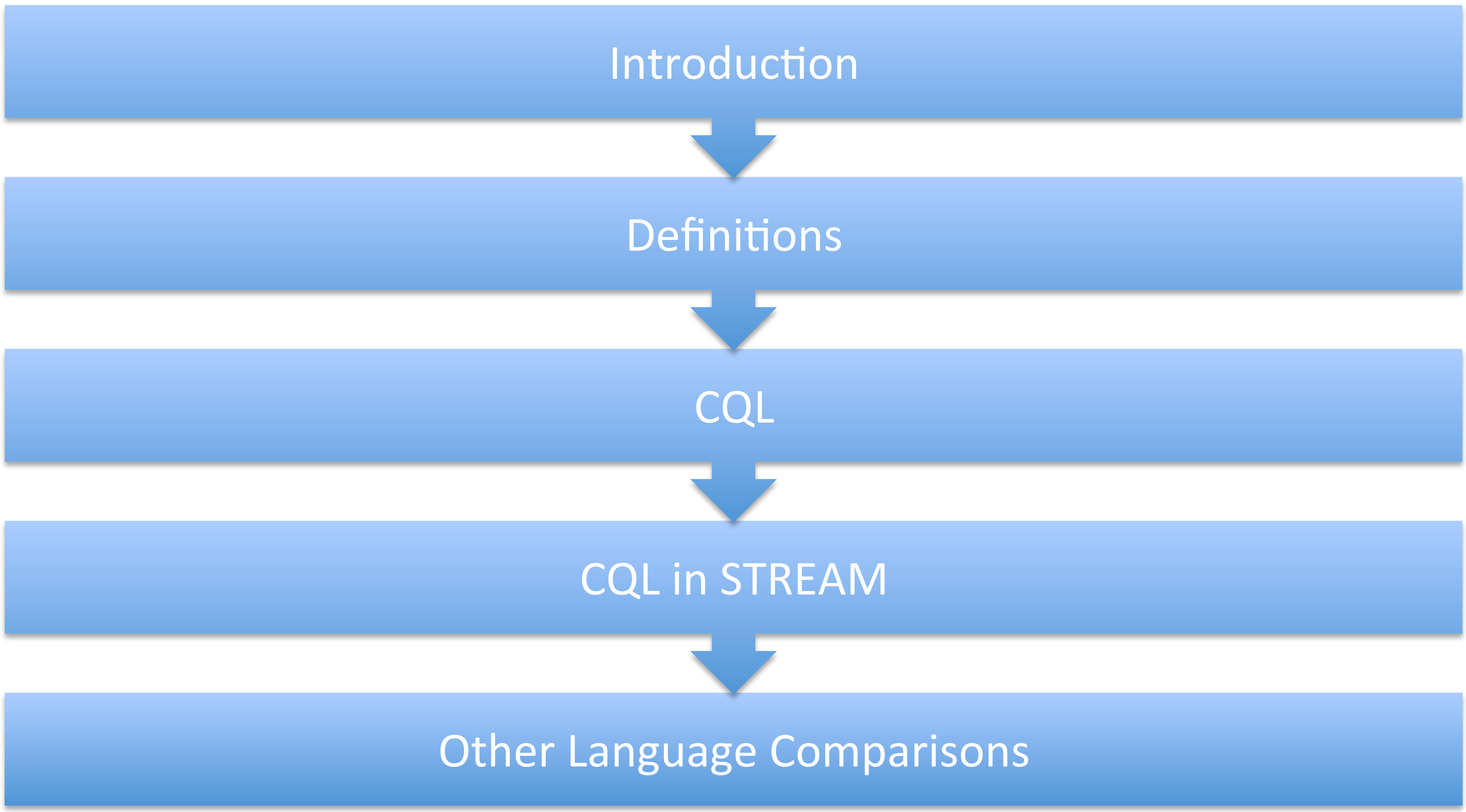
Query Execution Plans Matter Too! – Goals

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.*

A Roadmap


Introduction
Definitions
CQL
CQL in STREAM
Other Language Comparisons

The technical stuff.

DEFINITIONS

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

<u>Continuous Semantics</u>

- 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* τ s, there are 2 possibilities:
  1. The outermost operator in *Q* is relation-to-stream. The result of *Q at* time τ is *S* (the produced stream) *up to* τ*.*
  2. The outermost operator in *Q* is stream-to-relation or relation-to-relation. The result of *Q at* time τ is $R(\tau)$ (the produced relation).
- Time only "advances" from τ from (τ – 1) when all inputs up to τ – 1 have been processed.

What we're all here for.

CQL

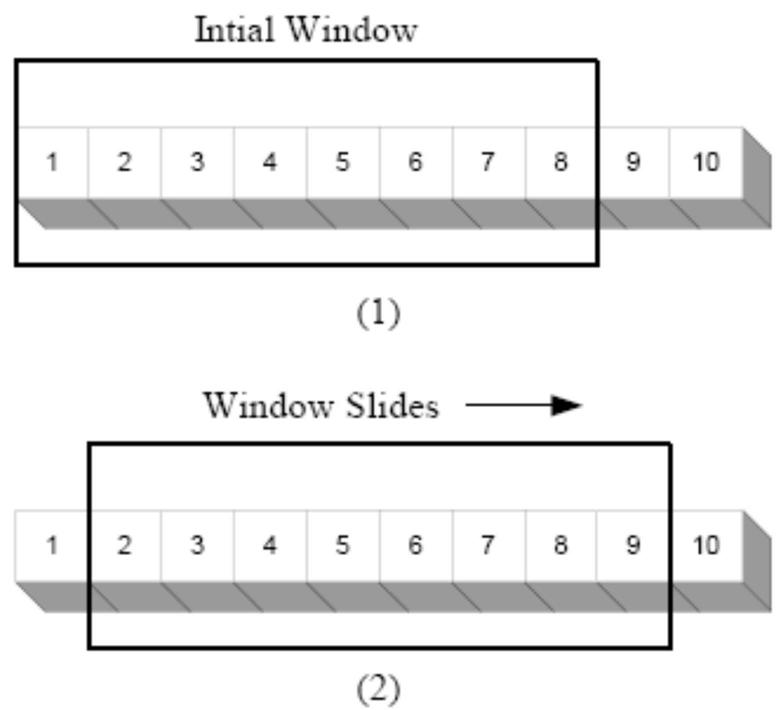
Operators: Stream-to-Relation

- 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, … , Ak Rows N]”



Operators: Relation-to-Relation

- Derived from traditional SQL.

Operators: Relation-to-Stream

- 3 operators:
  - Istream ("insert stream")

    $$\texttt{Istream}(R) = \bigcup_{\tau \geq 0} ((R(\tau) - R(\tau - 1)) \times \{\tau\})$$

  - Dstream ("delete stream")

    $$\texttt{Dstream}(R) = \bigcup_{\tau > 0} ((R(\tau - 1) - R(\tau)) \times \{\tau\})$$

  - Rstream ("relation stream")

    $$\texttt{Rstream}(R) = \bigcup_{\tau \geq 0} (R(\tau) \times \{\tau\})$$

Operators: Example

Relation-to-Stream

Stream-to-Relation

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

Relation-to-Relation

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

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

is equivalent to

SELECT L
FROM S [Range T]
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

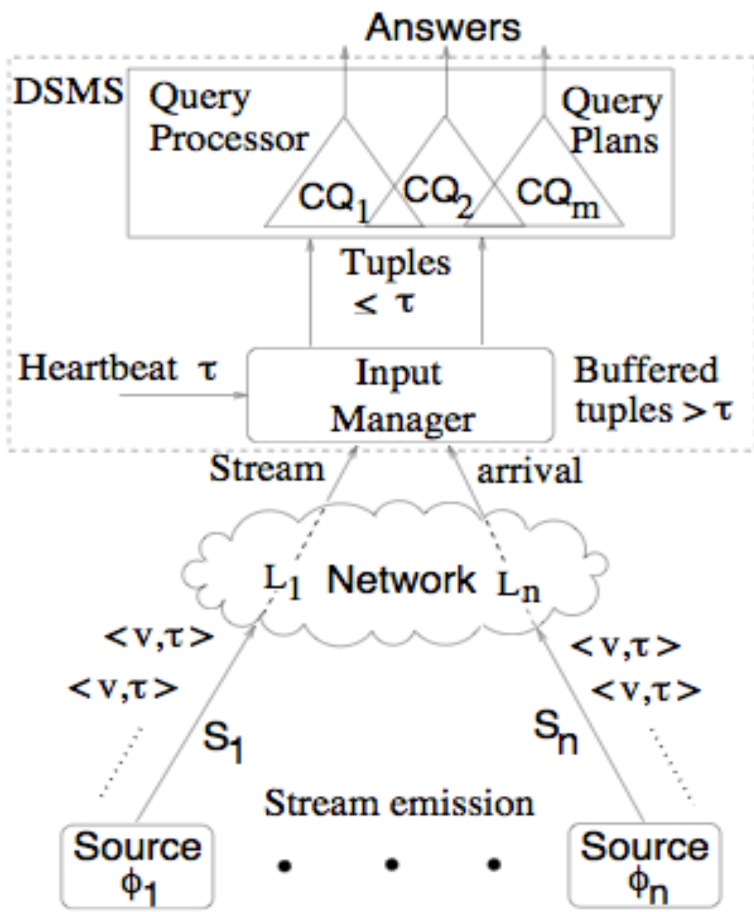
Time Management

- 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



Hot and STREAM-y.

CQL IMPLEMENTATION IN STREAM

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

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

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

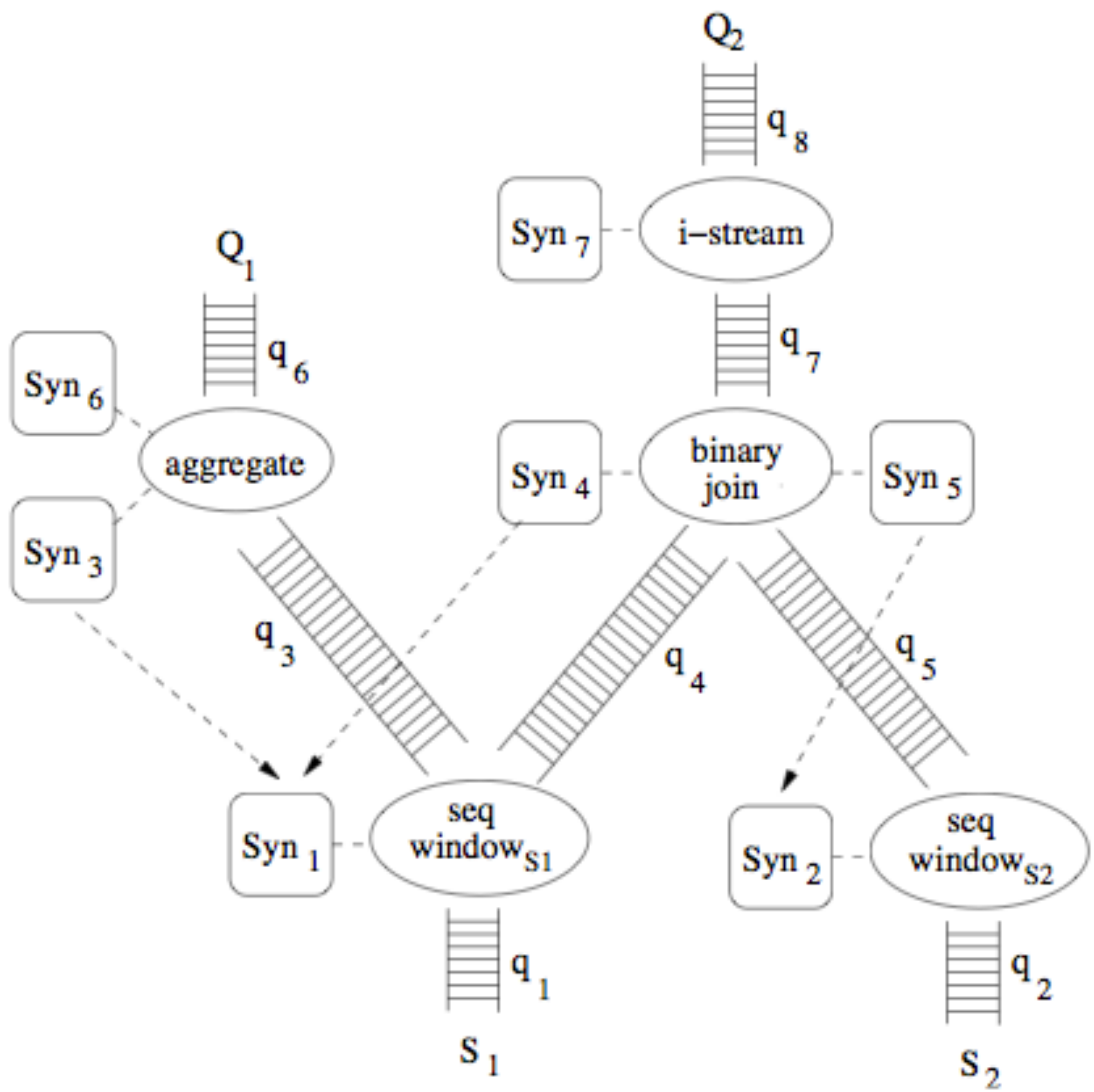
Q2:

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.

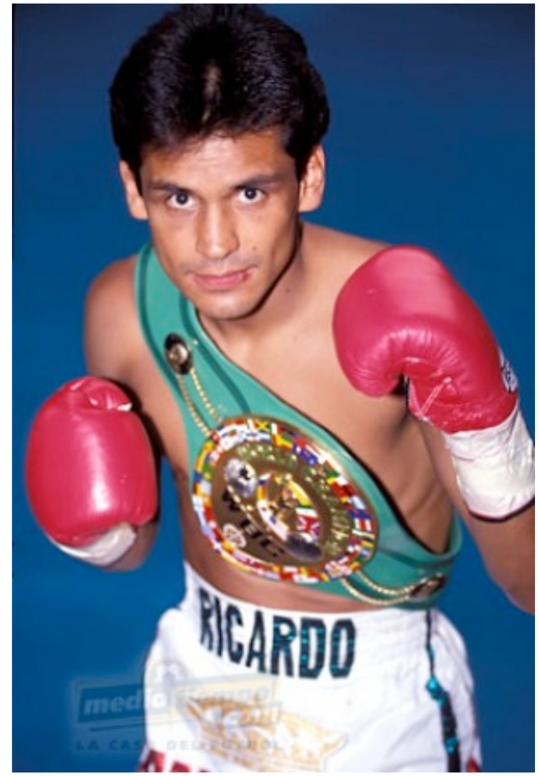
Aurora

- 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.ellenfinkelstein.com/pptblog/wordpress/wp-content/uploads/2011/the-beginning-road-sign.jpg
- StartFragment EndFragment
- http://medcitynews.com/wp-content/uploads/Black-Box-Art.png
- StartFragment EndFragment
- http://www.harborfreight.com/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/i/m/image_16157.jpg
- StartFragment EndFragment
- http://bethesdalutherancommunities.org/view.image?Id=2216
- StartFragment EndFragment
- http://wiki.treck.com/images/d/d4/Fig1.39_Sliding_Window_Protocol.gif
- StartFragment EndFragment
- U. Srivastava and J. Widom. Flexible time management in data stream systems. Page 264
- StartFragment EndFragment
- 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
- StartFragment EndFragment
- http://28.media.tumblr.com/tumblr_le900rkVhp1qzexono1_500.png
- https://40.media.tumblr.com/3290de4ee413685c60f08dc775310524/tumblr_mydnbyvVBr1rz41fxo2_400.jpg