

# Databases

Lecture 14

Data Streams

## Data Processing in Traditional DBMSs

- classical DBMSs answer the needs of traditional business applications
- finite data sets
- users execute queries on the database when necessary
- *one-shot (one-time)* query
  - executed on the current instance of the data (entirely stored)
  - finite time interval
  - specific to traditional DBMSs
- *human-active, DBMS-passive (HADP) model*
  - database - passive repository
  - users execute queries on the database when necessary

## Data Processing in Traditional DBMSs

- triggers
  - *second-class citizens*
- only the current state of the data is important
  - current data values are easy to obtain, whereas previous values can be painstakingly extracted from the log
- queries provide exact answers
- applications don't have real-time requirements

# Data Streams

- in a range of applications, data cannot be efficiently managed with a classical DBMS, as information takes the form of the so-called *data streams*
- e.g., astronomy, meteorology, seismology, financial services, e-commerce, etc.
- *data stream*
  - temporal sequence of values produced by a data source
  - potentially infinite
  - data arriving on the stream is associated with temporal values, i.e., *timestamps*

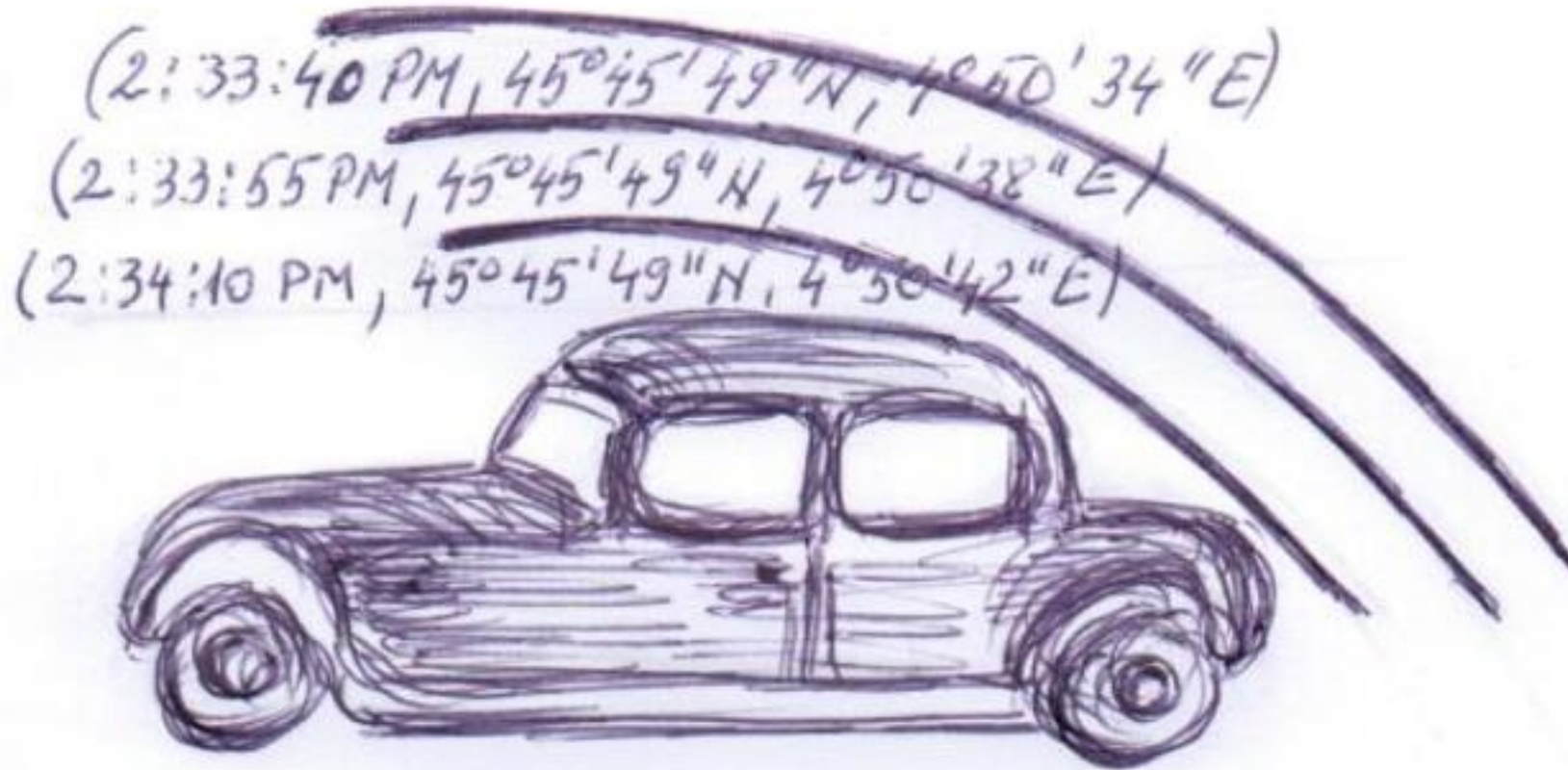
# Data Streams

- examples
  - a sequence of values provided by a temperature sensor
  - a sequence of GPS coordinates emitted by a car as it runs on a highway
  - a sequence of values representing a patient's heart rate and blood pressure
- time - common element in the examples above
- *event*
  - elementary unit of information that arrives on a data stream (similar to a record in relational databases); synonyms in this lecture, unless otherwise noted - *tuple*, *element*
- systems discussed in this lecture – structured data streams

# Data Streams

- *data source*
  - a device that provides a stream of values over time, in a digital format (a temperature sensor, a GPS device, a device that monitors a patient's heart, etc.)

# Data Streams



- 3 tuples on a stream of coordinates produced by the GPS device of a car
- the GPS emits the current location of the car (latitude and longitude) every 15 seconds

## Data Stream Monitoring Applications

- *monitoring applications*
  - applications that scan data streams, process incoming values, and compute the desired result
- e.g., military applications, financial analysis applications, variable tolling applications, etc.



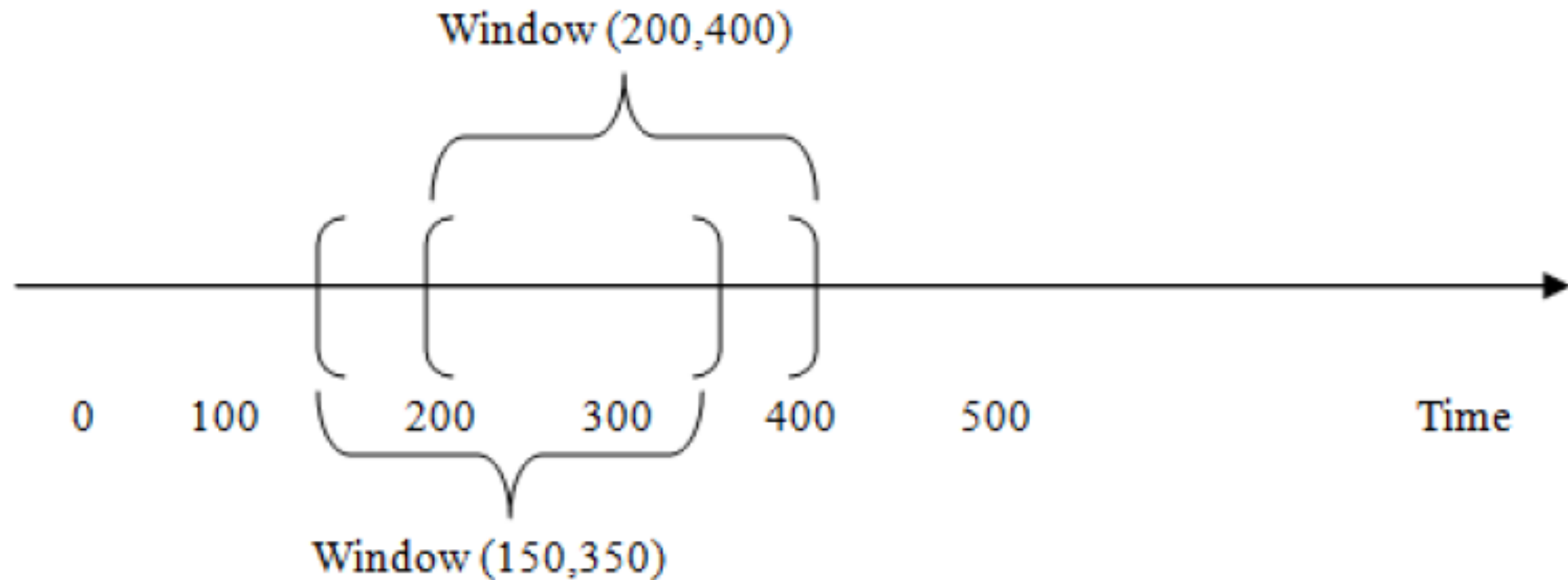
# Window-Based Processing Model

- data streams
  - potentially infinite
  - high data rates
- traditional DBMSs
  - vast storage space, secondary memory
- systems that process streams
  - usually rely on the main memory
- storing all the data - impossible
- data arriving on a stream
  - instantaneously processed, then eliminated
- evaluating queries on data streams
  - window-based model

## Window-Based Processing Model

- consider a temperature sensor in a refrigeration container; the user wants to be alerted whenever the temperature in the container exceeds a threshold 3 times in the last 10 minutes; it's enough to analyze the window of data that arrived on the stream in the previous 10 minutes; as time goes by and new tuples arrive on the stream, the window slides over the data in the stream
- *sliding window*
  - a contiguous portion of data from a stream
  - parameters
    - size - number of events / temporal instants
    - step size - number of events / temporal instants

# Window-Based Processing Model



- sliding window
  - size = 200 timestamps
  - step size = 50 timestamps

## Continuous Queries

- perpetually running queries, continuously producing results, while being fed with data from one or several streams
- provide real-time results, as required by many monitoring applications
  - e.g., variable tolling app that computes highway tolls based on dynamic factors such as accident proximity or traffic congestion
    - a driver must be alerted in real time whenever a new toll is issued for his or her car
    - providing this answer later in the future would be of no use
  - e.g., nuclear plant management

## Continuous Queries

- share similarities with views / triggers
  - materialized views change as the underlying tables change
  - condition statements from triggers
- one could add a large number of triggers to a DBMS and perform continuous processing in a traditional, although enhanced context
- the literature shows that a classical DBMS doesn't scale well past a certain number of triggers, whereas a monitoring application could easily track hundreds of streams with a large number of running continuous queries

## Continuous Queries

- continuous processing paradigm
  - *DBMS-active, human-passive (DAHP)*
  - database – active role
  - user – passive role

## Data Stream Management Systems

- the number of data sources providing monitored streams can grow significantly
- stream rates can be uniform, but data can also arrive in bursts (e.g., a stream of clicks from the website of a company when a new product is launched)
- the number of continuous queries / monitored data streams can also fluctuate considerably
- the complexity of the running queries can vary over time
- as system resources are limited, the system can become overloaded and unable to provide real-time results
- traditional DBMSs cannot tackle these challenges, being unable to efficiently manage data streams; dedicated systems, that use various strategies to handle such problems, are being used instead

## Data Stream Management Systems

- dedicated systems can execute continuous queries, while meeting the requirements of monitoring applications
- *Data Stream Management System*
  - system that processes streams of data in a perpetual manner, by running continuous queries
  - built around a query processing engine, which performs data manipulation operations
- academic prototypes
  - STREAM, Aurora, Borealis, etc.
- commercial systems
  - Azure Stream Analytics



# Data Stream Management Systems

- experimental results
  - a stream processing engine surpasses a traditional data processing engine in terms of performance when processing continuous and one-shot queries on streams and traditional data sets for a monitoring application

# Classical Databases Versus Data Streams

- classical DBMSs
  - permanent elements
    - data
  - temporary elements
    - queries
- DSMSs
  - permanent elements
    - continuous queries
  - transient elements
    - data arriving on streams

## STREAM - STandard stREam datA Manager

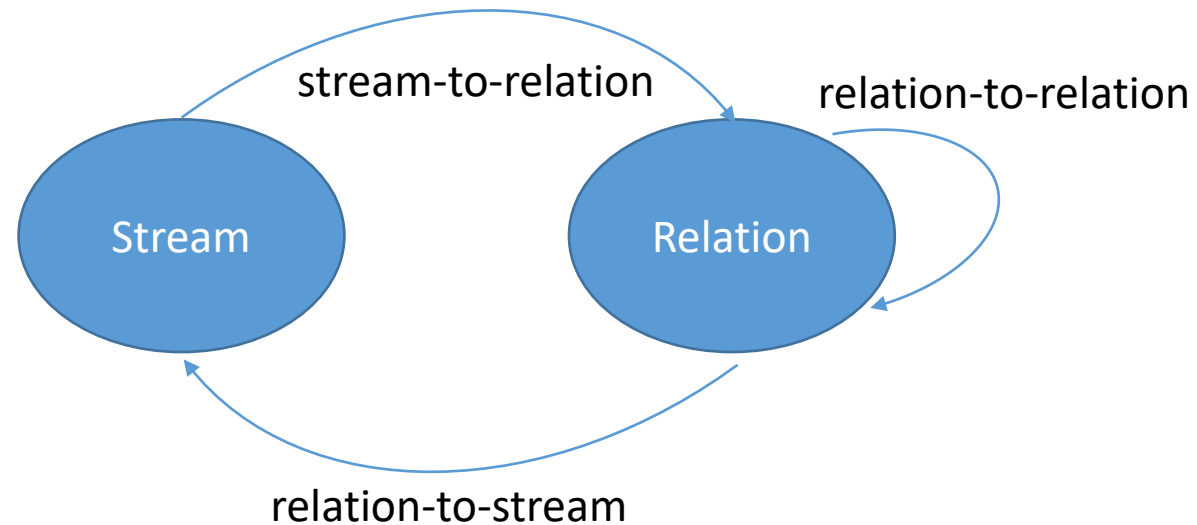
- DSMS prototype developed at Stanford
- objective
  - study data management and query processing in monitoring apps
- continuous queries on streams / stored data sets
- formal abstract semantics for continuous queries
- concrete declarative language, i.e., the Continuous Query Language (similar to SQL)

## STREAM - abstract semantics

- 2 data types
  - streams and relations
- discrete, ordered time domain  $T$ 
  - a timestamp  $t$  - a temporal moment from  $T$
  - $\{0, 1, \dots\}$
- data stream  $S$ 
  - unbounded multiset of tuple-timestamp pairs  $\langle s, t \rangle$
  - fixed schema, named attributes
- relation  $R$ 
  - time-varying multiset of tuples
  - $R(t)$  - instantaneous relation (i.e., the multiset of tuples at time  $t$ )
  - fixed schema, named attributes

# STREAM - abstract semantics

- 3 classes of operators
  - relation-to-relation
  - stream-to-relation
  - relation-to-stream



## STREAM - abstract semantics

- *relation-to-relation* operator
  - takes one or several input relations and produces an output relation
- *stream-to-relation* operator
  - takes an input stream and produces an output relation
- *relation-to-stream* operator
  - takes an input relation and produces an output stream
- stream-to-stream operators can be defined using the 3 classes of operators from the semantics
- operator classes
  - black box components
  - the semantics depends on the generic properties of each class, not on the operators' implementations

# STREAM - Continuous Query Language (CQL)

- minor extension of SQL
- defined by instantiating operators in the abstract semantics
- relation-to-relation operators
  - SQL constructs that transform several relations into a single relation
  - select, project, union, except, intersect, aggregate, etc.
  - $O_r$  - traditional relational operator over instantaneous input relations  $R_1, \dots, R_n$   
=> corresponding relation-to-relation operator in CQL  $O_c$  produces the time-varying relation  $R$ ; at timestamp  $t$ :  $R(t) = O_r(R_1(t), \dots, R_n(t))$

# STREAM - Continuous Query Language

- stream-to-relation operators
  - extract a sliding window from a stream
  - window-specification language derived from SQL-99
- sliding window - 3 types
  - tuple-based sliding window
  - time-based sliding window
  - partitioned sliding window



# STREAM - Continuous Query Language

- tuple-based sliding window
  - contains the last  $N$  tuples from the stream
  - $S$  - stream,  $N$  - positive integer
  - $S[\text{Rows } N]$  produces a relation  $R$
  - at time  $t$ ,  $R(t)$  contains the  $N$  tuples that arrived on  $S$  and have the largest timestamps  $\leq t$
- special case
  - $N = \infty$ 
    - $S[\text{Rows Unbounded}]$  - append-only window

# STREAM - Continuous Query Language

- time-based sliding window
  - $S$  - stream,  $t_i$  - temporal interval
  - $S[\text{Range } t_i]$  produces a relation  $R$
  - at time  $t$ ,  $R(t)$  contains the tuples that arrived on  $S$  and have the timestamps between  $t - t_i$  and  $t$
- special cases
  - $t_i = 0$ 
    - i.e., the tuples on  $S$  with timestamp =  $t$
    - $S[\text{Now}]$
  - $t_i = \infty$ 
    - tuples obtained from the elements of  $S$  up to  $t$
    - $S[\text{Range Unbounded}]$

## STREAM - Continuous Query Language

- time-based sliding window
  - e.g., CarStream(CarID, Speed, Position, Direction, Road)
    - CarStream[Range 60 seconds]
    - CarStream[Now]
    - CarStream[Range Unbounded]

## STREAM - Continuous Query Language

- partitioned sliding window
  - stream  $S$ ,  $N$  - positive integer,  $\{A_1, \dots, A_k\}$  - attributes in  $S$
  - $S[\text{Partition By } A_1, \dots, A_k \text{ Rows } N]$
  - logically partition  $S$  into substreams based on specified attributes
  - compute a tuple-based sliding window of size  $N$  on each substream
  - compute the union of resulting windows to produce the output relation
- e.g.,  $\text{CarStream}(\text{CarID}, \text{Speed}, \text{Position}, \text{Direction}, \text{Road})$ 
  - $\text{CarStream}[\text{Partition By CarID Rows } 1]$

# STREAM - Continuous Query Language

- relation-to-stream operators
- Istream (insert stream)
  - applied to a relation  $R$ , it contains  $\langle s, t \rangle$  whenever  $s$  is in  $R(t) - R(t-1)$  ( $s$  is added to  $R$  at time  $t$ )
- Dstream (delete stream)
  - applied to a relation  $R$ , it contains  $\langle s, t \rangle$  whenever  $s$  is in  $R(t-1) - R(t)$  ( $s$  is removed from  $R$  at time  $t$ )
- Rstream (relation stream)
  - applied to a relation  $R$ , it contains  $\langle s, t \rangle$  whenever  $s$  is in  $R(t)$  (every current tuple in  $R$  is streamed at every time instant)

## STREAM - Continuous Query Language

- example CQL queries
- `CarStream(CarID, Speed, Position, Direction, Road)`
- at any given time, display the set of active cars (i.e., having transmitted a position report in the past 60 seconds)

```
SELECT DISTINCT CarID
```

```
FROM CarStream[Range 60 Seconds]
```

- the result is a relation

## STREAM - Continuous Query Language

- example CQL queries
- windowed join of 2 streams

```
SELECT *  
FROM S1 [ROWS 200], S2 [RANGE 5 Minutes]  
WHERE S1.Attr = S2.Attr AND S1.Attr < 500
```

- result = relation
- at every temporal instant  $t$ , the result contains the join (on *Attr*) of the last 200 tuples of  $S1$  with the tuples that have arrived on  $S2$  in the past 5 minutes; only tuples with  $Attr < 500$  are part of the result

## STREAM - Continuous Query Language

- example CQL queries
- stream containing new *Attr* values, as they appear in the join

```
SELECT Istream(S1.Attr)
FROM S1 [ROWS 200], S2 [RANGE 5 Minutes]
WHERE S1.Attr = S2.Attr AND S1.Attr < 500
```

- result = stream



## STREAM - Continuous Query Language

- example CQL queries
- `SegmentSpeedStream(segment, road, dir, speed, ...)`
- display, at any given time, the set of congested road segments, i.e., segments for which the avg. speed of cars in the past 10 min is below 70 kph

```
SELECT segment, road, dir
FROM SegmentSpeedStream[Range 10 Minutes]
GROUP BY segment, road, dir
HAVING AVG(speed) < 70
```

## STREAM - Continuous Query Language

- example CQL queries
- SegmentSpeedStream(CarID, Segment, Road, Dir, Speed)

- display the current road segments for all active vehicles

```
SELECT DISTINCT U.CarID, U.Segment, U.Road, U.Dir  
FROM SegmentSpeedStream[Range 60 Seconds] A,  
     SegmentSpeedStream[Partition By CarID Rows 1] U  
WHERE A.CarID = U.CarID
```

- result = relation

## STREAM – execution plans

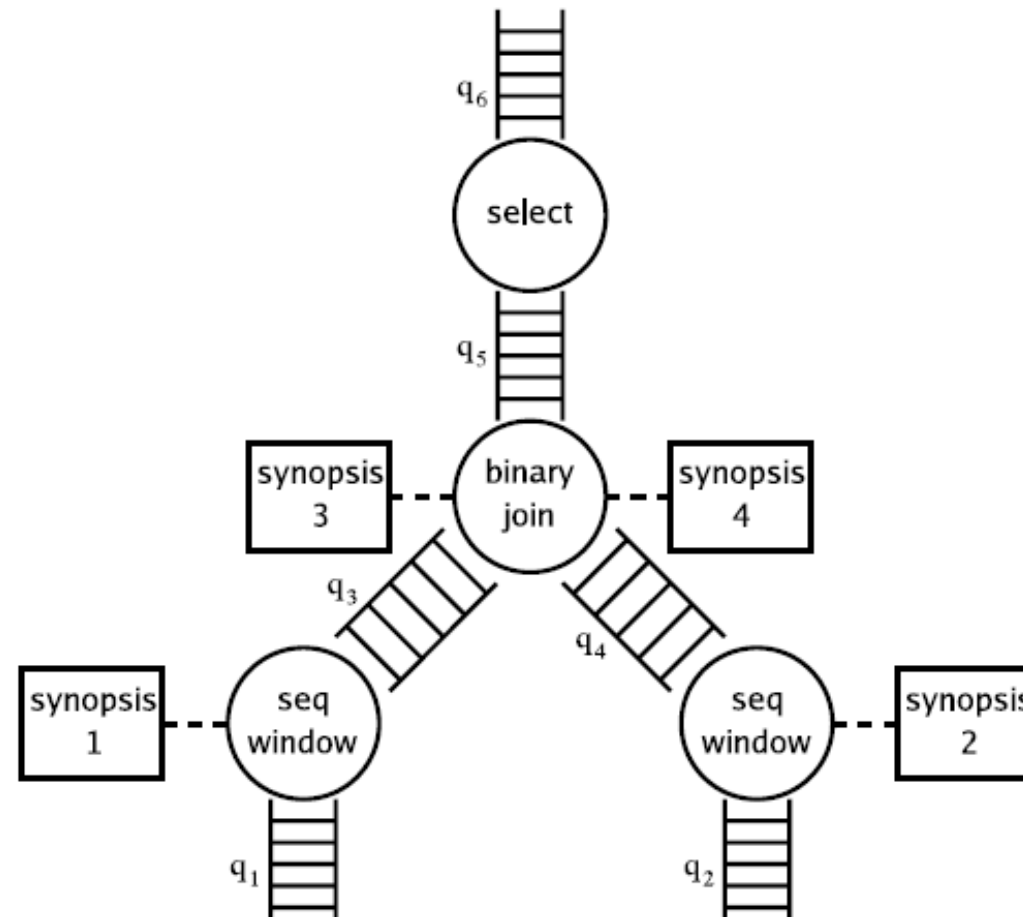
- when registered with the system, a CQL query is compiled into a physical query plan
- plan
  - tree structure
  - operators
    - perform the processing
  - inter-operator queues
    - hold elements as they move between operators
  - synopses attached to operators
    - store state when necessary, e.g., a join of 2 windows
    - operators that don't require a synopsis
      - selection, duplicate-preserving union, etc.
  - leaves - inputs; root operator - computes the result of the query

# STREAM – execution plans

SELECT \*

FROM S1 [ROWS 200], S2 [RANGE 5 Minutes]

WHERE S1.Attr = S2.Attr AND S1.Attr < 500



STREAM - maybe in 2 years from now (Master's Programmes) :)

- sharing data & computation within and across execution plans
- exploiting stream constraints - ordering, clustering, etc.
- load-shedding
- etc.

## References

- Daniel J. Abadi, Donald Carney, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul and Stanley B. Zdonik. Aurora: a new model and architecture for data stream management. The VLDB Journal, 12(2):120–139, 2003
- A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava and J. Widom. STREAM: The Stanford Data Stream Management System. Technical Report, Stanford InfoLab, 2004
- Arvind Arasu, Shivnath Babu and Jennifer Widom. The CQL continuous query language: Semantic foundations and query execution. The VLDB Journal-Raport tehnic, 15(2):121–142, 2006
- A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryzkina, M. Stonebreaker and R. Tibbetts. Linear Road: A Stream Data Management Benchmark. In VLDB'04, Proceedings of The Thirtieth International Conference on Very Large Data Bases, pages 480–491, 2004

## References

- Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani and Jennifer Widom. Models and Issues in Data Stream Systems. In *PODS*, pages 1–16, 2002
- Y. Gripay, F. Laforest, F. Lesueur, N. Lumineau, J.-M. Petit, V.-M. Scuturici, S. Sebahi, S. Surdu, Colistrack: Testbed For A Pervasive Environment Management System, Proceedings of The 15th International Conference on Extending Database Technology (EDBT 2012), 574-577, 2012
- \*\*\* Azure Stream Analytics - technical documentation, <https://azure.microsoft.com/en-us/services/stream-analytics/>