Databases

Lecture 14

Query Optimization in Relational Databases. Evaluating Relational Algebra Operators

Data Streams

Sabina S. CS

SQL Statements Execution

- client application - SQL statement execution request
  - for any query - minimum response time
- statement execution - stages:
  - client: generate SQL statement (non-procedural language), send it to server
  - server:
    - analyze SQL statement (syntactically)
    - translate statement into an internal form (relational algebra expression)
    - transform internal form into an optimal form
    - generate a procedural execution plan
    - evaluate procedural plan, send result to client

- the following operators are necessary in the querying process:
  - selection: $\sigma_C(R)$
  - projection: $\pi_\alpha(R)$
  - cross-product: $R_1 \times R_2$
  - union: $R_1 \cup R_2$
  - set-difference: $R_1 - R_2$
  - intersection: $R_1 \cap R_2$
  - theta join: $R_1 \otimes_\Theta R_2$
  - natural join: $R_1 * R_2$
  - left outer join: $R_1 \ltimes_C R_2$
  - right outer join: $R_1 \rtimes_C R_2$
  - full outer join: $R_1 \bowtie_C R_2$
  - left semi join: $R_1 \rhd R_2$
  - right semi join: $R_1 \lhd R_2$
  - division: $R_1 \div R_2$
  - duplicate elimination: $\delta(R)$
  - sorting: $S_{\{list\}}(R)$
  - grouping: $\gamma_{\{list1\}\, group\ by\ \{list2\}}(R)$

- an SQL query can be written in multiple ways
- example for a relational database
- primary keys are underlined, foreign keys are written in blue

  programs[<u>id</u>, pname, pdescription]

  groups[<u>id</u>, program, yearofstudy, gdescription]

  students[<u>cnp</u>, lastname, firstname, sgroup, gpa, addr, email]
- query: find students (lastname, firstname, year of study, program name, gpa) in a given program (e.g., with id = 2, can be a parameter), with a gpa >= 9 (can be a parameter):

a)

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM students st, groups gr, programs pr
WHERE st.sgroup = gr.id AND gr.program = pr.id
  AND program = 2 and gpa >= 9
```

b)

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM (students st INNER JOIN groups gr ON
  st.sgroup = gr.id)
  INNER JOIN programs pr ON gr.program = pr.id
WHERE program = 2 AND gpa >= 9
```

c)

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM
   (
    (SELECT lastname, firstname, sgroup, gpa
     FROM students
     WHERE gpa >= 9) st
    INNER JOIN
    (SELECT * FROM groups WHERE program = 2) gr
       ON st.sgroup = gr.id
   )
   INNER JOIN
   (SELECT id, pname FROM programs WHERE id = 2) pr
   ON gr.program = pr.id
```

- the previous query versions are equivalent (they provide the same answer)
- equivalent relational algebra expressions:

a.

$$\pi_\beta(\sigma_C(students \times groups \times programs))$$

b.

$$\pi_\beta(\sigma_{C1}((students \otimes_{C2} groups) \otimes_{C3} programs))$$

c.

$$\pi_\beta(((\pi_{\beta 1}(\sigma_{C2}(students))) \otimes_{C3} (\sigma_{C4}(groups))) \otimes_{C5} (\pi_{\beta 2}(\sigma_{C6}(programs))))$$

- an evaluation tree can be constructed for a relational algebra expression
- problems:
    - which version is better?
    - when generating the execution plan:
        - which parameters are optimized?
        - what information is required?
    - what can the optimizer (DBMS component) do?

Relational Algebra Operators - Evaluation

- operands for relational operators:
  - database tables (can have attached indexes)
  - temporary tables (obtained by evaluating some relational operators)
- several evaluation algorithms can be used for a relational algebra operator
- when generating the execution plan:
  - choose the algorithm with the lowest complexity (for the current database context); take into account data from the system catalog, statistical information

- a join can be defined as a cross-product followed by a selection
- joins arise more often in practice than cross-products
- in general, the result of a cross-product is much larger than the result of a join
- it's important to implement the join without materializing the underlying cross-product, by applying selections and projections as soon as possible, and materializing only the subset of the cross-product that will appear in the result of the join

Cross Join

- this algorithm is used to evaluate a cross-product:
  - R CROSS JOIN S
  - R INNER JOIN S ON C (C evaluates to TRUE)
  - SELECT ... FROM R, S ..., no join condition between R and S
- b_R, b_S
  - the number of blocks storing R and S, respectively
- m, n
  - the number of blocks from R and S that can simultaneously appear in the main memory (there are m+n buffers for the 2 tables)

Cross Join

- the following algorithm can be used to generate the cross-product $\{(r, s) \mid r \in R, s \in S\}$:
- `for every group of max. m blocks in R:`
  - `read the group of blocks from R into main memory; let` M_1 `be the set of records in these blocks`
  - `for every group of max. n blocks in S:`
    - `read the group of blocks from S into main memory; let` M_2 `be the set of records in these blocks`
    - `for every r ∈` M_1`:`
      - `for every s ∈` M_2`: add (r, s) to the result`

Cross Join

- algorithm complexity: total number of read blocks (from the 2 tables):

$$b_R + \left\lceil \frac{b_R}{m} \right\rceil * b_S \tag{1}$$

(number of blocks in R; for every group of max. m blocks in R, read S)

- to minimize this value, m should be maximized (the other operands are constants); one buffer can be used for S (so n = 1), while the remaining space can be used for R (m max.)
- switch the 2 relations (in the algorithm and when computing the complexity) => complexity:

$$b_S + \left\lceil \frac{b_S}{n} \right\rceil * b_R \tag{2}$$

- choose better version
- obs.: if b_R <= m or b_S <= n => complexity $b_R + b_S$

Nested Loops Join

- the Cross Join algorithm can be used to evaluate a join between 2 tables
- for every element (r, s) in the cross-product, evaluate the condition in the join operator
- elements (r, s) that don't meet the join condition are eliminated

Indexed Nested Loops Join

- this algorithm is used to evaluate $R \otimes_C S$, where C ≡ (R.A=S.B), and there is an index on A (in R) or on B (in S)
- in the algorithm description below, we assume there is an index on column B in table S
- `for every block in R:`
  - `read the block into main memory; let M be the set of records in the block`
  - `for every r ∈ M:`
    - `determine v =` $\pi_A(r)$
    - `use the index on B in S to determine records s ∈ S with value v for B; for every such record s, the pair (r,s) is added to the result`
- obs.: depending on the type of index - at most 1 / multiple matching records in S

Merge Join

- this algorithm is used to evaluate $R \otimes_C S$, where C ≡ (R.A=S.B), and there are no indexes on A (in R) and B (in S)
- sort R and S on the columns used in the join: R on A, S on B
- scan obtained tables; let r in R and s in S be 2 current records
  - if r.A = s.B: add (r', s') to the result; r' is in the set of all consecutive records in R with A = r.A, similarly for s' in S; next(r); next(s) (get a record with the next value for A and B)
  - if r.A < s.B: next(r) (determine record in sorted R with the next value for A)
  - if r.A > s.B: next(s) (determine record in sorted S with the next value for B)

Relational Algebra Equivalences

- SQL statement - transformed into a relational algebra expression (based on a set of transformation rules for the clauses that appear in the statement)
- transform relational expression (such that the evaluation algorithm has a lower complexity)
- certain transformation rules are used (mathematical properties of the relational operators)

* $\sigma_C(\pi_\alpha(R)) = \pi_\alpha(\sigma_C(R))$

- selection reduces the number of records for projection; in the second expression, the projection operator analyzes fewer records
- optimization - algorithm that evaluates both operators in a single pass of R

* perform one pass instead of 2:

$$\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C1\ \mathrm{AND}\ C2}(R)$$

* replace cross-product and selection by condition join (a number of condition join algorithms don't evaluate the cross-product):

$$\sigma_C(R \times S) = R \otimes_C S$$

, where C - join condition between R and S

* R and S - compatible schemas:

$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$
$$\sigma_C(R \cap S) = \sigma_C(R) \cap \sigma_C(S)$$
$$\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$$

* $\sigma_C(R \times S)$

particular cases:

- C contains only attributes from R:

$$\sigma_C(R \times S) = \sigma_C(R) \times S$$

- C = C1 AND C2, C1 contains only attributes from R, C2 - only attributes from S:

$$\sigma_{C1 \text{ AND } C2}(R \times S) = \sigma_{C1}(R) \times \sigma_{C2}(S)$$

- C = C1 AND C2, C2 - join condition between R and S:

$$\sigma_{C1 \text{ AND } C2}(R \times S) = \sigma_{C1}(R \otimes_{C2} S)$$

* $\pi_{\alpha}(R \cup S) = \pi_{\alpha}(R) \cup \pi_{\alpha}(S)$

* $\pi_{\alpha}(R \otimes_{C} S) = \pi_{\alpha}(\pi_{\alpha 1}(R) \otimes_{C} \pi_{\alpha 2}(S))$

- $\alpha 1$: attributes in R that appear in α or C
- $\alpha 2$: attributes in S that appear in α or C

* associativity and commutativity for some relational operators

- associativity and commutativity for ∪ and ∩
- associativity for the cross-product and the natural join
- "equivalent" results (same records, but different column order) when commuting operands in × and certain join operators
  - R × S = S × R – when using the Cross Join algorithm, the order of the data sources is important

* transitivity of some relational operators for the join operators - additional filters could be applied before the join:

- (A>B AND B>3) ≡ (A>B AND B>3 AND A>3)
- example: A is in R, B is in S:

$$R \otimes_{A>B \text{ AND } B>3} S = (\sigma_{A>3}(R)) \otimes_{A>B} (\sigma_{B>3}(S))$$

- (A=B AND B=3) ≡ (A=B AND B=3 AND A=3)
- example: A is in R, B is in S:

$$R \otimes_{A=B \text{ AND } B=3} S = (\sigma_{A=3}(R)) \otimes_{A=B} (\sigma_{B=3}(S))$$

* evaluating $\sigma_C(R)$, where C ≡ $(R.A \in \delta(\pi_{\{B\}}(S)))$; avoid evaluating C for every record of R; the initial evaluation is equivalent to:

$$R \otimes_{R.A=S.B} (\delta(\pi_{\{B\}}(S)))$$

- consider again the query described on the database:
  programs[id, pname, pdescription]
  groups[id, program, yearofstudy, gdescription]
  students[cnp, lastname, firstname, sgroup, gpa, addr, email]
- query: find students (lastname, firstname, year of study, program name, gpa) in a given program (e.g., with id = 2), with a gpa >= 9:

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM students, groups, programs
WHERE students.sgroup = groups.id AND
  groups.program = programs.id AND
  program = 2 and gpa >= 9
```

- denote by:

C $\equiv$ (students.sgroup = groups.id AND groups.program = programs.id AND program = 2 and gpa >= 9)

β = {lastname, firstname, yearofstudy, pname, gpa} – attributes in the SELECT clause

- the corresponding relational expression:

$$\pi_{\beta}(\sigma_C(students \times groups \times programs))$$

* carry out the following transformations, using previously discussed rules:

- associativity for ×:

$$students \times groups \times programs = (students \times groups) \times programs$$ or

$$students \times groups \times programs = students \times (groups \times programs)$$

- commute σ with × (a particular case); use the transitivity of the equality operator:

(groups.program = programs.id AND program = 2)

≡ (groups.program = programs.id AND program = 2 AND programs.id = 2)

students.sgroup = groups.id	AND groups.program = programs.id	AND program = 2	AND gpa >= 9	AND programs.id = 2
C1	C2	C3	C4	C5

$$\sigma_C(students \times groups \times programs) =$$
$$\sigma_{C1\ AND\ C2}((\sigma_{C4}(students) \times \sigma_{C3}(groups)) \times \sigma_{C5}(programs))$$ or
$$\sigma_{C1\ AND\ C2}(\sigma_{C4}(students) \times (\sigma_{C3}(groups) \times \sigma_{C5}(programs)))$$

- replace selection and cross-product with condition join:

$$= ((\sigma_{C4}(students)) \otimes_{C1} (\sigma_{C3}(groups))) \otimes_{C2} (\sigma_{C5}(programs))$$

or

$$= (\sigma_{C4}(students)) \otimes_{C1} ((\sigma_{C3}(groups)) \otimes_{C2} (\sigma_{C5}(programs)))$$

- choose a version based on statistical information from the database; we consider the first version:

$$\Rightarrow e = \pi_{\beta}(((\sigma_{C4}(students)) \otimes_{C1} (\sigma_{C3}(groups))) \otimes_{C2} (\sigma_{C5}(programs)))$$

- commute π with join:

$\beta 1$ = {lastname, firstname, gpa, sgroup} - useful for β and join

$\beta 2$ = {id, program, yearofstudy} - useful for β and join

$\beta 3$ = {id, pname} - useful for β and join

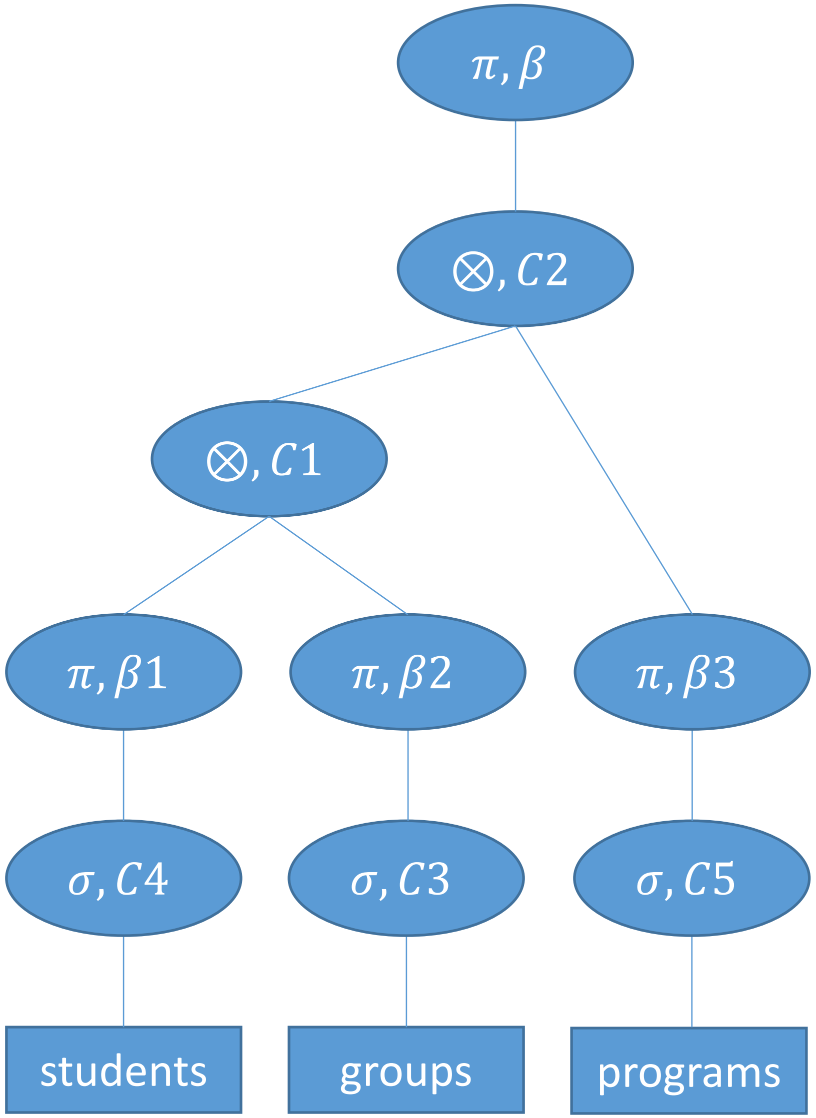
$$e = \pi_{\beta}(((\pi_{\beta 1}(\sigma_{C4}(students))) \otimes_{C1} (\pi_{\beta 2}(\sigma_{C3}(groups)))) \otimes_{C2} (\pi_{\beta 3}(\sigma_{C5}(programs))))$$

- the last expression corresponds to the statement:

```
SELECT lastname, firstname, yearofstudy, pname, gpa
FROM
  (
   (SELECT lastname, firstname, gpa, sgroup FROM students WHERE gpa >= 9) st
   INNER JOIN
   (SELECT id, program, yearofstudy FROM groups WHERE program = 2) gr
      ON st.sgroup = gr.id
  )
  INNER JOIN
  (SELECT id, pname FROM programs WHERE programs.id = 2) pr
  ON gr.program = pr.id
```

- an evaluation tree can be constructed for the last version of the relational algebra expression

- using information from the system catalog and possibly statistical information, an execution plan can be generated from the last version of the expression; every relational operator is replaced by an evaluation algorithm



* Let P, Q, R be 3 relations with schemas P[PID, P1, P2, P3], Q[QID, Q1, Q2, Q3, Q4, Q5], R[RID, R1, R2, R3], and E an expression in the relational algebra:

$$E = \pi_{\{P2,\ Q2,\ Q4,\ R3\}}(\sigma_{PID\ =\ Q1\ AND\ QID\ =\ R2\ AND\ P3='Bilbo'\ AND\ Q5\ =\ 100\ AND\ R1=7}(P \times Q \times R))$$

Optimize E and draw the evaluation tree for the optimized version of the expression.

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

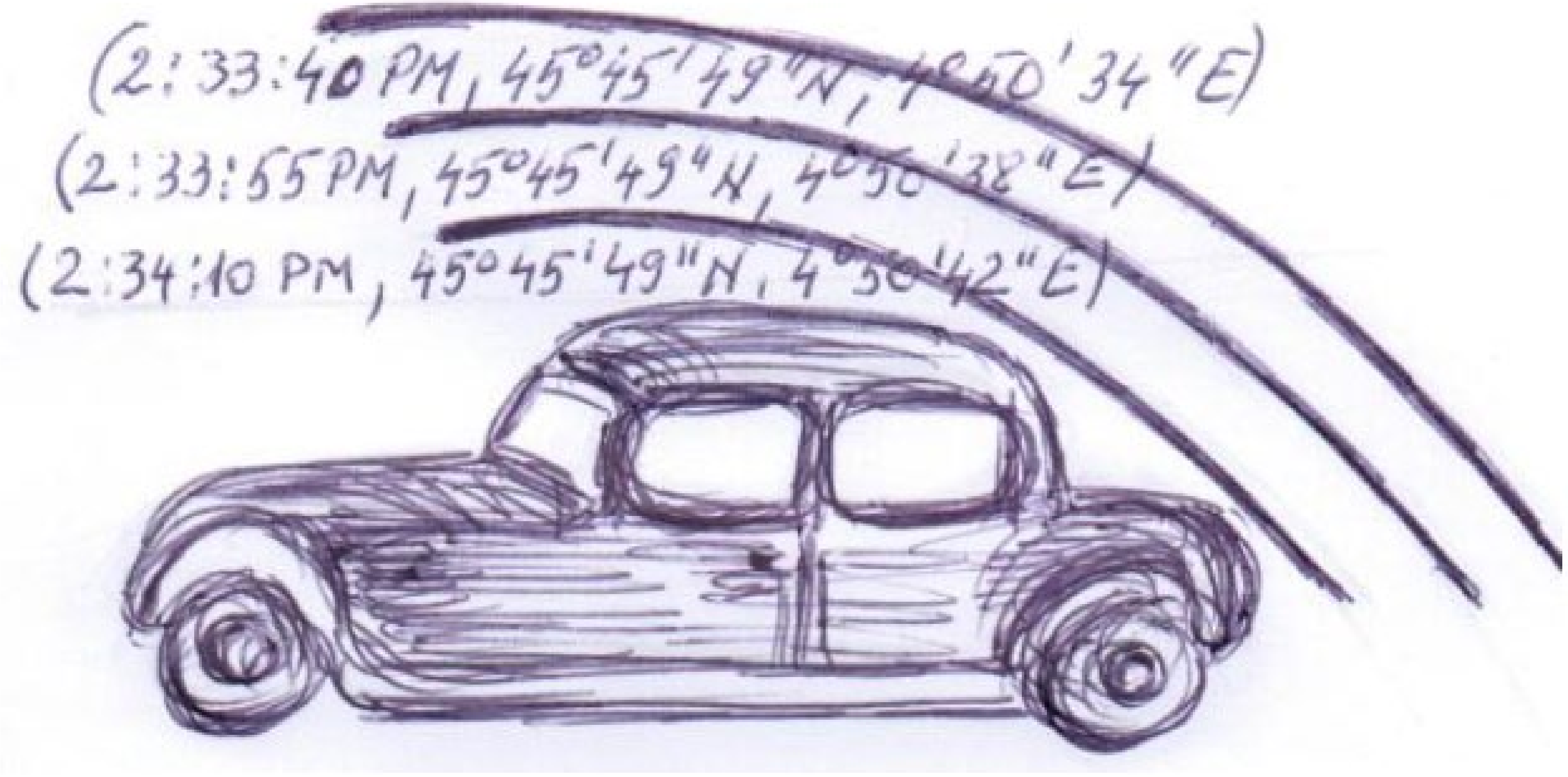
Data Streams

- 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*
- 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

Data Streams

- 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 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.

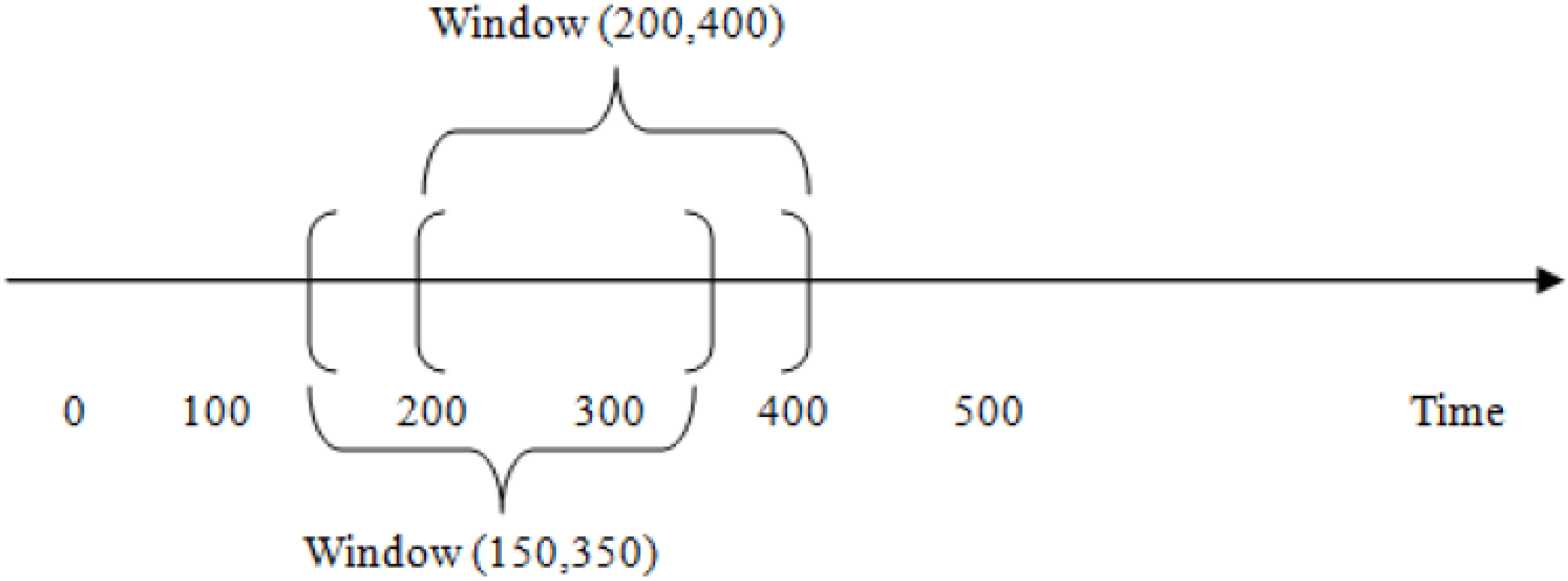
Window-Based Processing Model

- 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 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

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 - STandord stREam datA Manager

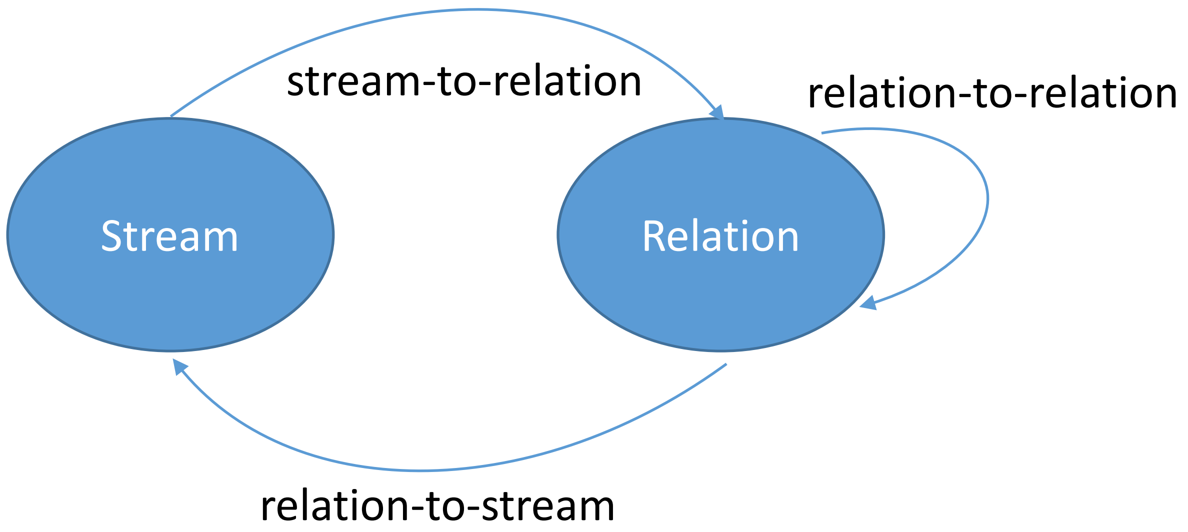
- DSMS prototype developed at Stanford
- objective
  - study data management and query processing in monitoring apps
- continuous queries on streams / stored data sets
- formal abstract semantics for continuous queries
- concrete declarative language, i.e., the Continuous Query Language (similar to SQL)

STREAM - abstract semantics

- 2 data types
    - streams and relations
- discrete, ordered time domain T
    - a timestamp t - a temporal moment from T
    - {0, 1, ...}
- data stream S
    - unbounded multiset of tuple-timestamp pairs <s, t>
    - 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.
- 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[Rows N] produces a relation R
  - at time t, R(t) contains the N tuples that arrived on S and have the largest timestamps <= t
  - special case
    - N = ∞
      - S[Rows Unbounded] - append-only window

STREAM - Continuous Query Language

- time-based sliding window
  - S - stream, ti - temporal interval
  - S[Range ti] produces a relation R
  - at time t, R(t) contains the tuples that arrived on S and have the timestamps between t-ti and t
  - special cases
    - ti = 0
      - i.e., the tuples on S with timestamp = t
      - S[Now]
    - ti = ∞
      - S[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

- relation-to-stream operators
- Istream (insert stream)
  - applied to a relation R, it contains <s, t> 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 <s, t> 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 <s, t> 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 *S*1 with the tuples that arrived on S2 in the past 5 minutes; only tuples with *Attr* < 500 are part of the result

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

- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Da03] DATE, C.J., An Introduction to Database Systems (8th Edition), Addison-Wesley, 2003
- [Si11] SILBERSCHATZ, A., KORTH, H., SUDARSHAN, S., Database System Concepts (6th Edition), McGraw-Hill, 2011
- [Kn76] KNUTH, D.E., Tratat de programare a calculatoarelor. Sortare și căutare. Ed. Tehnică, București, 1976
- [Ga09] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., Database Systems: The Complete Book (2nd Edition), Pearson Education, 2009

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. Ryvkina, 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/