SEQUENCE QUERY PROCESSING

***Praveen Seshadri, Miron Livny, Raghu Ramakrishnan* (CS Department, University of Wisconsin-Madison, WI)**

Sedat Behar and Yevgeny Ioffe

REFERENCES

- Praveen Sheshadri, *Management of Sequence Data* (PhD Thesis)
- R.Ramakrishnan, M.Chend, M.Livny, P.Sheshadri. *What's Next? Sequence Queries*
- P.Sheshadri, M.Livny, R.Ramakrishnan. *SEQ: A Model for Sequence Databases*
- H.Gunadhi, A.Segev. *Event-join optimization in temporal relational databases*

WHY? (Difficulties with expressing Sequences)

- Relational Databases:
  - Data collections treated as sets not sequences
- Difficult to evaluate in SQL
  - Data model does not help evaluation
- Hard to optimize:
  - scan two sequences in lock-step

Query: Find the Volcano that caused an earthquake of magnitude 7.0 or greater

```
SELECT V.name
FROM Volcanos V, Earthquakes E
WHERE E.Strength > 7.0 AND
        E.time = (SELECT max(E1.time)
                  FROM Earthquakes E1
                  WHERE E1.time < V.time)
```

It is difficult and not efficient!

SEQUENCE MODEL

Definitions:

- **Record**: $<A_1:T_1, A_2:T_2,\ldots, A_n:T_n>$
- **Attribute**: data type (int, String…)
- **Type**: an instance in the type domain (Null associated)
- **Position**: location of an entry in the record
- **Position Ordering**: function that returns the position (index)
- **Type Domain**: $(T_1 \times T_2 \times \ldots \times T_n)$

Example of a Record:

<Name:String, HeartBeat:Int, BloodPres:Int …>

SEQUENCE MODEL

- Types of Sequences:

  BASE SEQUENCES: some positions map to some records

  CONSTANT SEQUENCES: every position maps to the same unique record
  Ex: Perfect Health (Mr.Perfect, 80, 80/120)

  DERIVED SEQUENCES: defined by a sequence operator

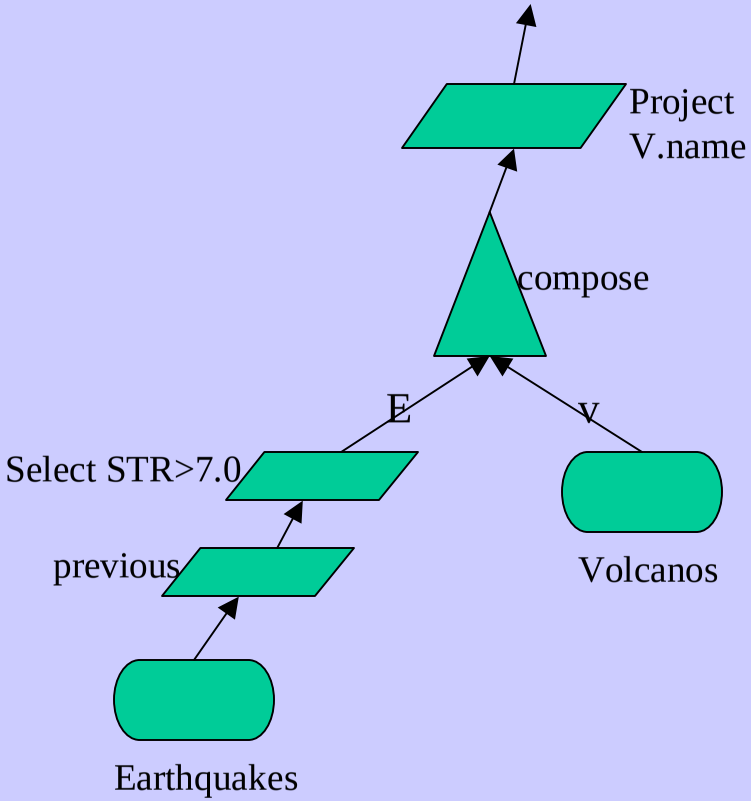
SEQUENCE MODEL - OPERATORS

- All operators are compositional: produce a single derived sequence from 2 sequences
- Simple Unary Operators:
  - ***Selection:*** *similar to Relational Databases*
  - ***Projection****: similar to Relational Databases*
  - ***Positional Offset****: shifts input sequence by offset*
  - ***Value Offset****: shifts the non-null entries by offset*
- Aggregate Unary Operators
  - ***agg_pos(i):*** selects set P of positions for each position **i**
  - ***agg_func:*** aggregate function over records in input stream at positions p in P
- Compose Operator (*positional join operator*)
  - binary operator composing records r1 and r2 of two input sequences at each position **i** (*Null exception*)

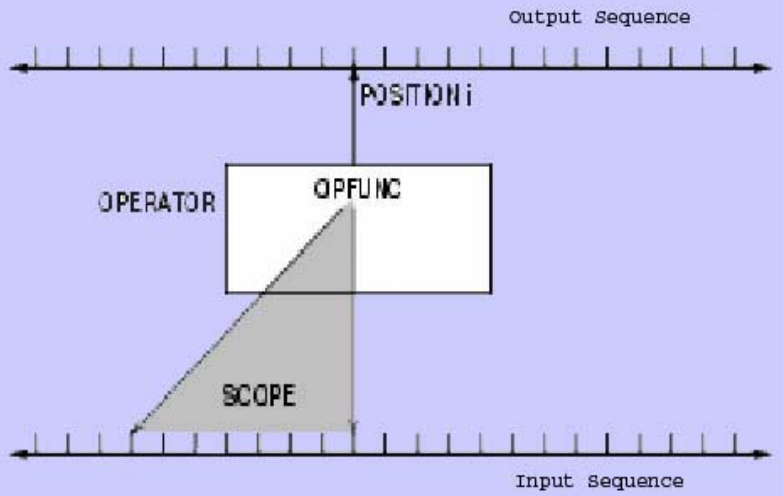
SEQUENCE MODEL

Sequence Queries:

- A sequence query is an acyclic graph of operators (just like Aurora!)
- Output of query is output sequence
- No output can be input to more than 1 operator → graph is a tree



SCOPE OF OPERATOR



- important for optimization and evaluation!
- Operator can be described by 2 functions:
  - *Scope*: defines the positions of input records to look at
  - *OpFunc*: an operator function that actually works with input records defined by Scope to define the output sequence

OPERATOR PROPERTIES

- Operator properties:
  - Scope size at position *i*
  - Scope sequentiality
  - Scope relativity at position *i*
- Complex operator
  - Is an acyclic composition of basic operators
  - Properties of basic operators determine its property (*fixed scope; sequential scope; relative scope*)

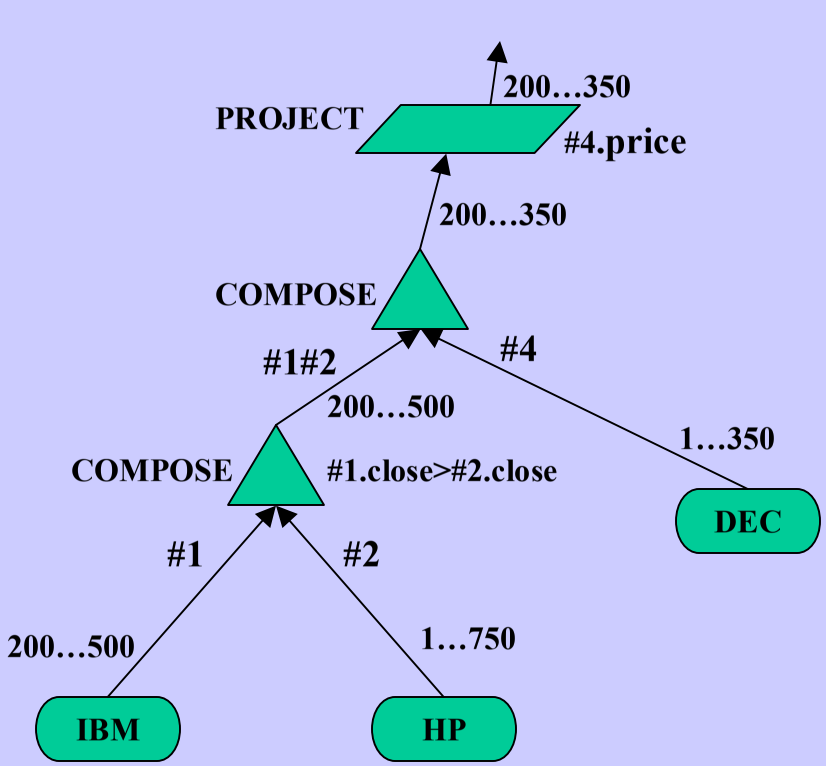
QUERY OPTIMIZATION - TRANSFORMATIONS

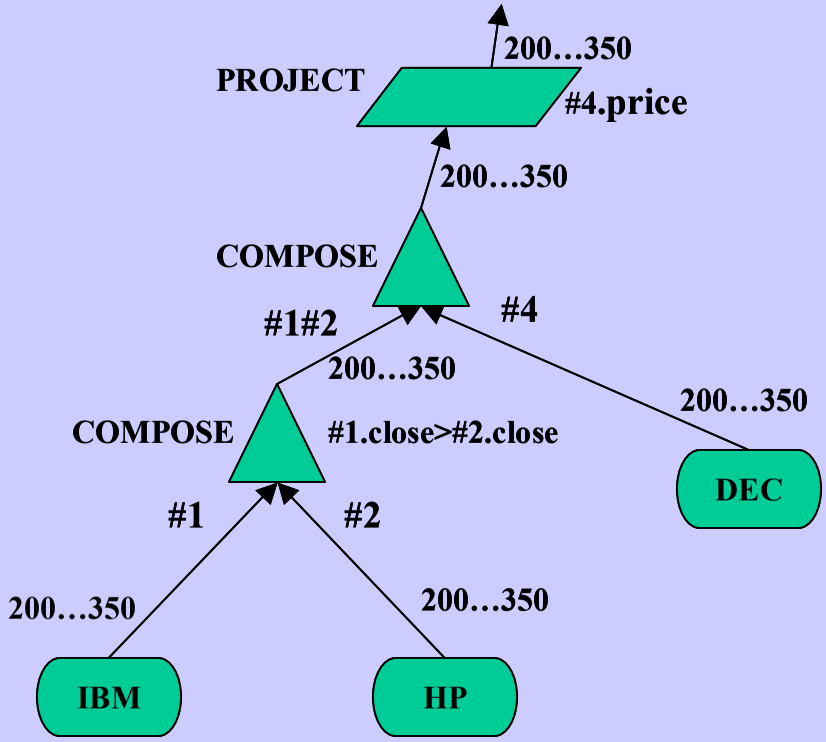
- Transform declarative query into equivalent one
- Two sequence queries are equivalent if both have same:
  - input sequences
  - scopes on input sequences
  - operator function
- Equivalence is independent of actual data in input sequences

QUERY OPTIMIZATION – TRANSFORMATIONS

- Alter a sub-query but not the entire graph
- Incorrect transformations
- Good idea to propagate selections, projections, and positional offsets as far down as possible
- Non-unit scope operators → break query into blocks

GLOBAL SPAN OPTIMIZATION

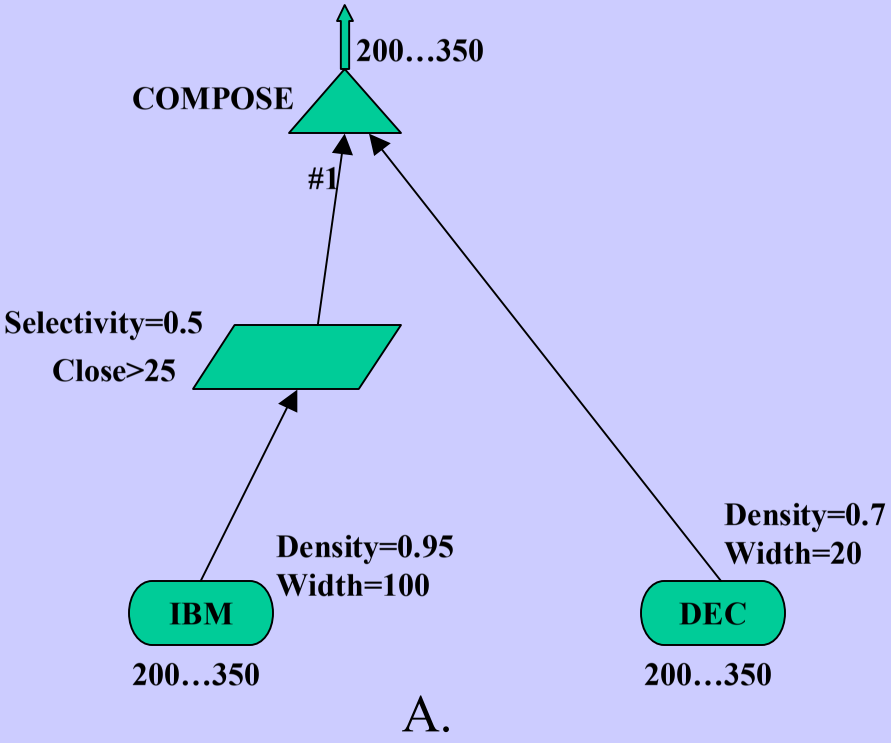


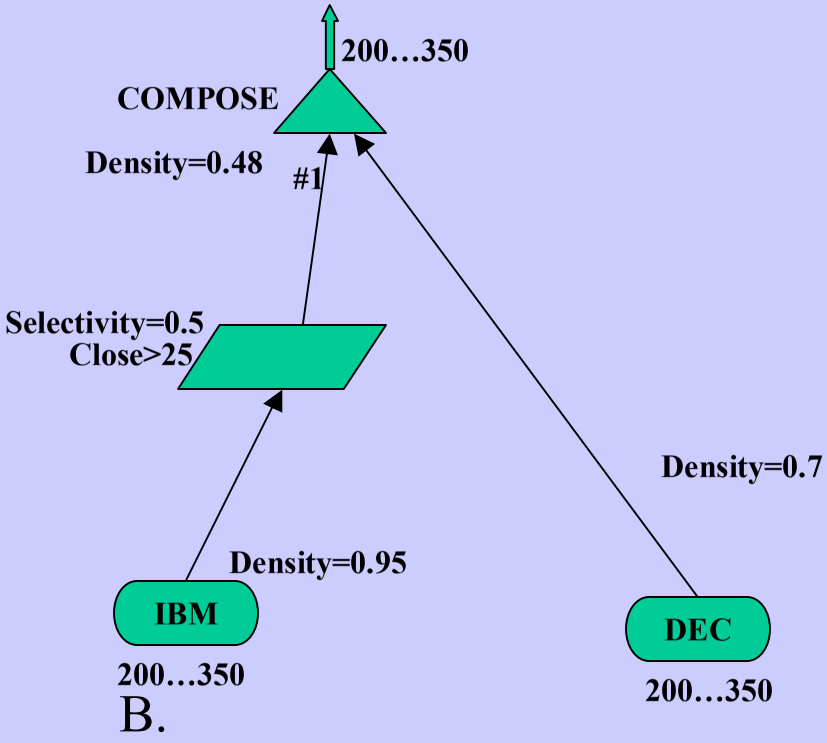


- reduce query processing costs by restricting the span of a sequence based on span of other sequences
  - → can modify span of output based on input and vice versa!

META-INFORMATION

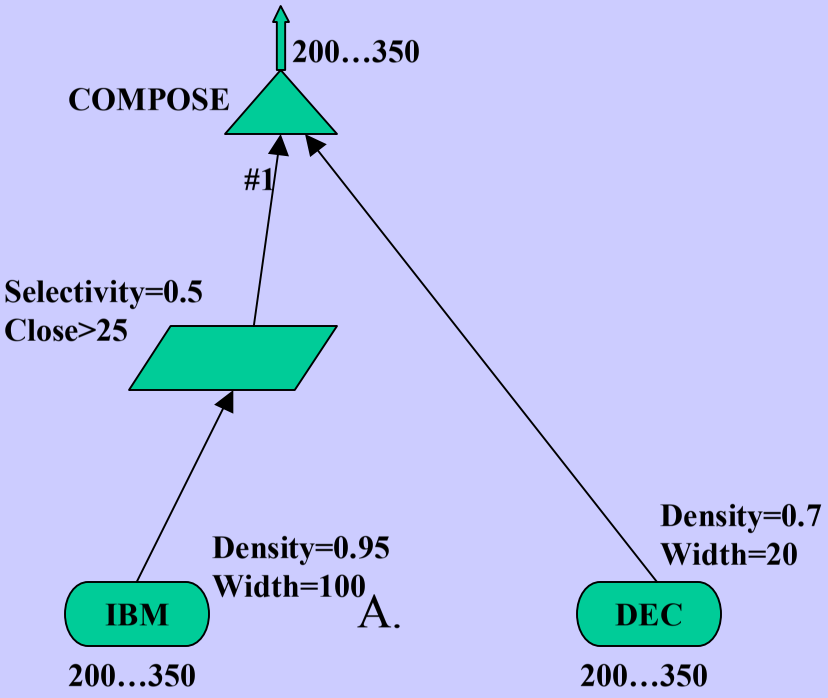
- Concepts: Span and Density

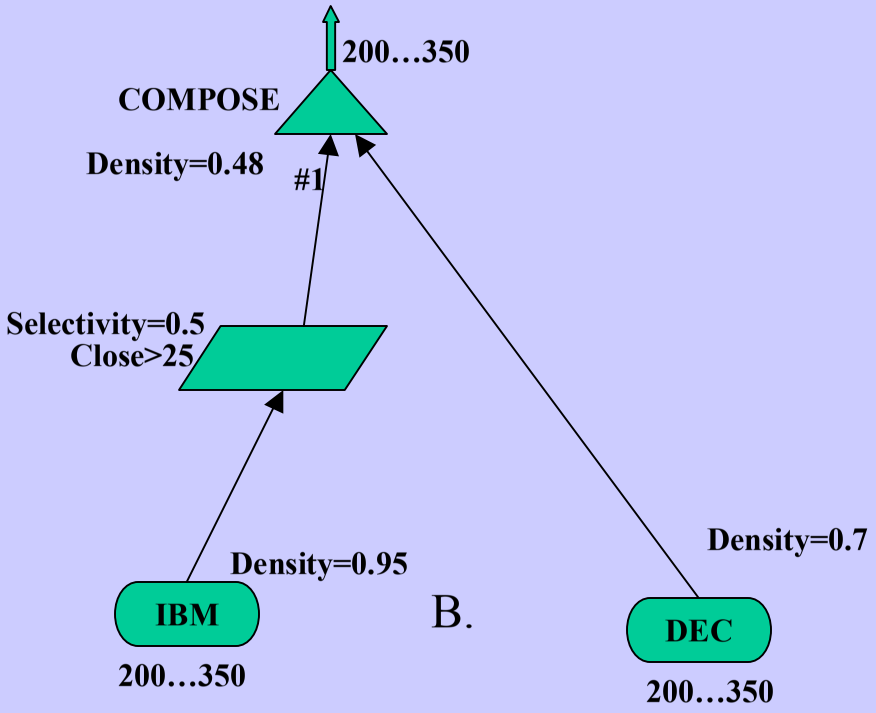




- Utilize span first then density to reduce the workload on join

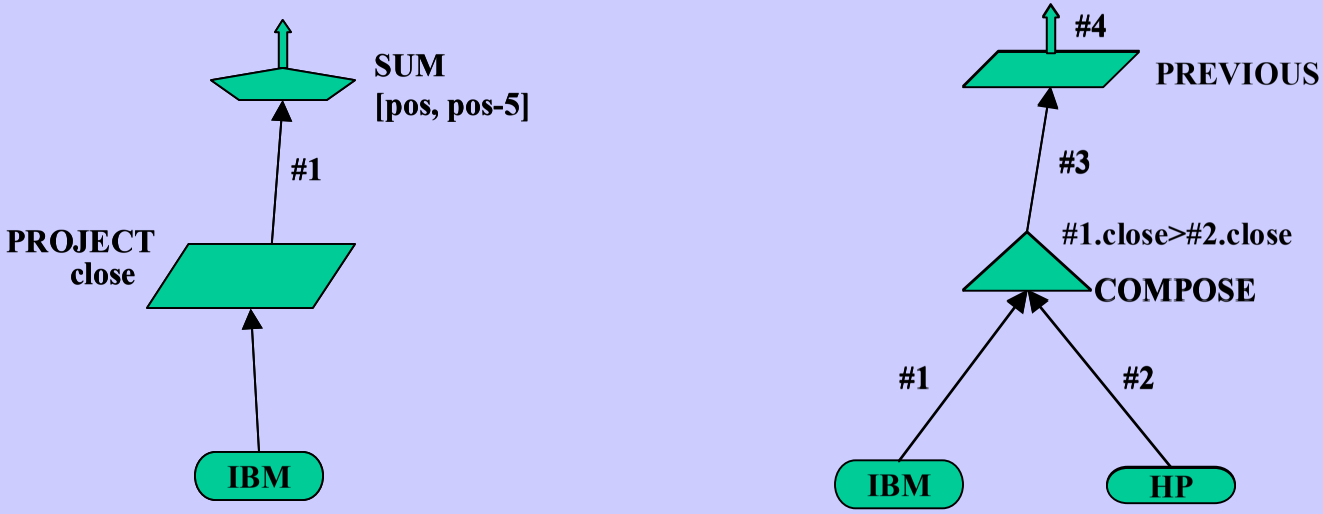
ACCESS MODES





- Join Strategy-A – two ways of doing it
  - analogous to NLJ
- Join Strategy-B: Stream both in lock-step
- "stream access" = get the **next** non-Null record
- "probed access" = get the record at a **specific** position

CACHING OF DERIVED SEQUENCES



- *Cache-Strategy-A:*
  - Cache last 6 values of sequence #1 (figure on left)
  - If scope is large or variable → may not be feasible to cache whole scope

- *Cache-Strategy-B (incremental cache strategy):*
  - Cache the value of #4 at previous position → then the record at some position p is either cached record at previous position OR it's the non-Null record from #3 at previous position

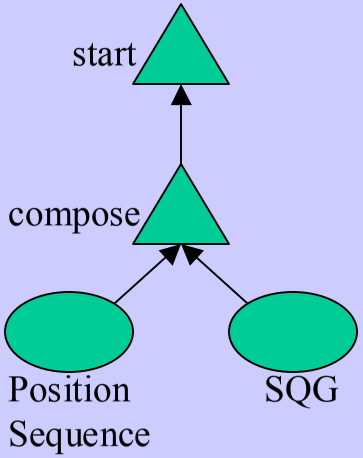
QUERY PLAN GENERATION

Sequences may be queried for
- specific positions
- range of positions

Plan Generation Algorithm:

SQG: signifies that there might be a derived sequence represented below, not necessarily a base sequence.

Start: initiates query evaluation by invoking stream access on its input



QUERY PLAN GENERATION

Step 1- Query Specification

Step 2- Meta-Information Propagation:
- *Bottom-up*
- *Top-down*

Step 3- Query Transformation

Step 4- Identification of Query Blocks

Step 5- Block-wise Plan Generation

Step 6- Plan Selection

QUERY PLAN GENERATION

Access costs to Base Sequences:

size of valid range, density of sequence, access paths available

Costs: Blocks with Non-Unit Scope: (Aggregate & Value Offset)

–If stream access,

- cost = stream cost of input + cost of storing in cache each record + cost of cache access + computational cost

–If probed access,

- cost = probed access cost × operator scope size

Costs: Blocks with Positional Joins:

Stream Access: Stream access on stream-1 and probe on stream-2; or converse or stream access on both streams

Probed Access: Access stream-1 in probed fashion and for every record join stream-2 in probed fashion

Algorithmic Analysis: left-deep trees, time and space complexity

Extensions

- To Model:
  General Sequences, Ordering Domains, Multiple Orderings, Sequence Groupings
- To Queries:
  Generalized Query Graph, Correlated Queries
- To Framework:
  Optimization Framework, Materialization of Derived Sequences, Optimizations on base sequences (sorting)

Related Work

- Compare to TS-based models
- Aurora, Stream-based systems

CONCLUSIONS/Things to remember

- Two important things:
  scope and query generation plans

Discussion Questions

- How can this algorithm extend (relate) to Aurora or Atlas?
- How is operator scope extended/implemented in CQL?