Optimization of Nonsequenced Queries using Log-Segmented Timestamps

Curtis Dyreson

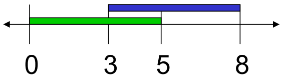
Utah State University, Logan, Utah, USA

Outline

- Timestamp representation
- Application to nonsequenced queries
- Evaluation
- Conclusion

Timestamp Representation

- An interval/period timestamp



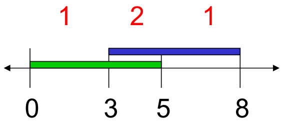
- Intervals are
  - [0, 3]
  - [5, 8]
- Features
  - Just *start* and *stop* times
  - Minimal information, smallest in terms of storage
  - What is an alternative and why would anyone do it differently?

Temporal Grouping and Aggregation

- Temporal data about football players

players

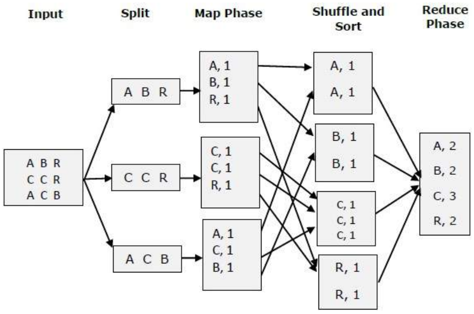
player	team	time
Wilson	ManU	[0-5]
Arnhelm	ManU	[3-8]
…	…	…
…	…	…



- How many players on each team *at the same time*?
  - Groups vary over time
  - Player belongs to potentially *n*^2 groups
  - Special aggregation techniques for temporal aggregation

Sequenced Aggregation in Map/Reduce

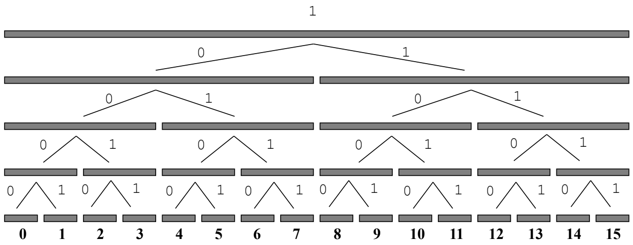
- Interval representation is bad in Map/Reduce



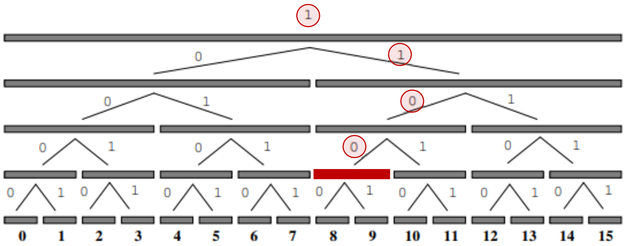
- Need a new kind of timestamp
- Curtis E. Dyreson: Using CouchDB to Compute Temporal Aggregates. *HPCC/SmartCity/DSS* 2016: 1131-1138

Log-segmented Timestamp

- Problem: can’t shard intervals
- Introduce log-segmented timestamp
- Partition timeline into pre-defined segments

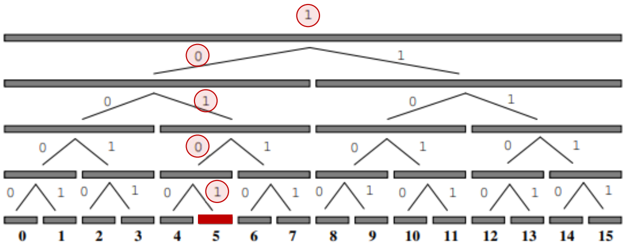


Log-segmented Example



Log-segmented label for period [8, 9] is **1100**

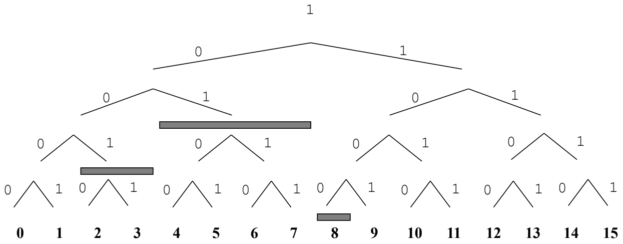
Log Segmented Example



Log-segmented label for period [5, 5] is **10101**

Convert Periods to Segments

- Period [2-8] is {1001, 101, 11000}



- Compact – $2\log_2(n)$ segments can represent any period

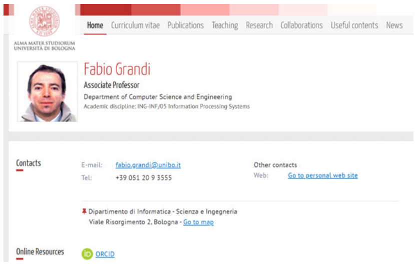
Log-segmented Sequenced Semantics

- Sequenced semantics for relational DBs
- Curtis E. Dyreson, M. A. Manazir Ahsan: Achieving a Sequenced, Relational Query Language with Log-Segmented Timestamps. TIME 2021: 14:1-14:13

Outline

- Timestamp representation
- Application to nonsequenced queries
- Evaluation
- Conclusion

Fabio Grandi E-mail



Fabio Grandi

Associate Professor

Department of Computer Science and Engineering

Academic discipline: ING-INF/05 Information Processing Systems

Contacts

E-mail: fabio.grandi@unibo.it

Tel: +39 051 20 9 3555

Other contacts

Web: Go to personal web site

Dipartimento di Informatica - Scienza e Ingegneria
Viale Risorgimento 2, Bologna - Go to map

Online Resources

ORCID

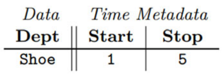
- Fabio*: Interesting idea, but what about log-segmented for nonsequenced queries, after all, nonsequenced is more important than sequenced*
- Me*: Yes, nonsequenced is important, but log-segmented timestamps don't improve nonsequenced*

Nonsequenced Semantics

- Most common temporal extension of a query language
- Nonsequenced: query has explicit temporal predicates and constructors
- Benefit
  - *Temporal* can be added to any DBMS
  - Layer, no DBMS modification

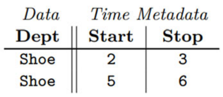
Example Nonsequenced Join

```
SELECT s.dept, OVERLAPS(r, s)
FROM tesco s, walmart r
WHERE r OVERLAPS s
```



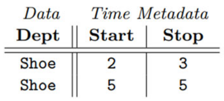
Data	*Time Metadata*	
Dept	**Start**	**Stop**
Shoe	1	5

(a) Relation Tesco



Data	*Time Metadata*	
Dept	**Start**	**Stop**
Shoe	2	3
Shoe	5	6

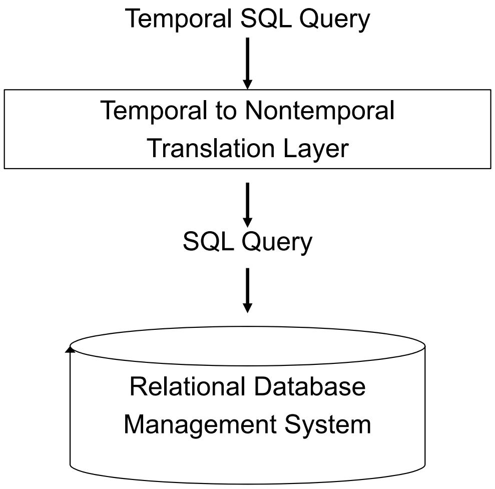
(b) Relation Walmart



Data	*Time Metadata*	
Dept	**Start**	**Stop**
Shoe	2	3
Shoe	5	5

(c) Result of the nonsequenced evaluation of the query in Figure 1.

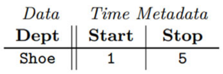
System Architecture



Temporal SQL Query
Temporal to Nontemporal Translation Layer
SQL Query
Relational Database Management System

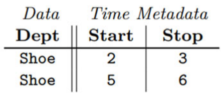
Example Nonsequenced Join

```
SELECT s.dept,
   GREATEST(r.time.start, s.time.start) AS start,
   LEAST(r.time.stop, s.time.stop) as stop
FROM tesco s, walmart r
WHERE ((r.start <= s.start AND s.start <= r.stop)
    OR (s.start <= r.start AND r.start <= s.stop))
```



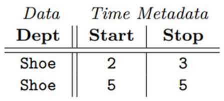
Data	*Time Metadata*	
Dept	**Start**	**Stop**
Shoe	1	5

(a) Relation Tesco



Data	*Time Metadata*	
Dept	**Start**	**Stop**
Shoe	2	3
Shoe	5	6

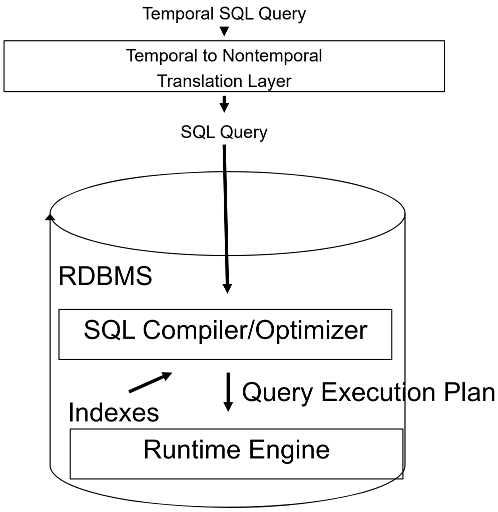
(b) Relation Walmart



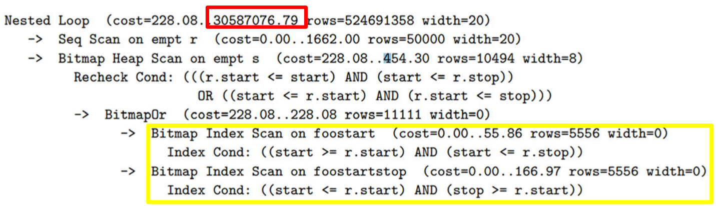
Data	*Time Metadata*	
Dept	**Start**	**Stop**
Shoe	2	3
Shoe	5	5

(c) Result of the nonsequenced evaluation of the query in Figure 1.

System Architecture



Query Execution Plan



```
Nested Loop  (cost=228.08..30587076.79 rows=524691358 width=20)
  ->  Seq Scan on empt r  (cost=0.00..1662.00 rows=50000 width=20)
  ->  Bitmap Heap Scan on empt s  (cost=228.08..454.30 rows=10494 width=8)
        Recheck Cond: (((r.start <= start) AND (start <= r.stop))
                      OR ((start <= r.start) AND (r.start <= stop)))
        ->  BitmapOr  (cost=228.08..228.08 rows=11111 width=0)
              -> Bitmap Index Scan on foostart  (cost=0.00..55.86 rows=5556 width=0)
                   Index Cond: ((start >= r.start) AND (start <= r.stop))
              -> Bitmap Index Scan on foostartstop  (cost=0.00..166.97 rows=5556 width=0)
                   Index Cond: ((start <= r.start) AND (stop >= r.start))
```

- Cost of query highlighted in red
- Note use of indexes highlighted in yellow
- We can lower cost of query from 30,587,076 to 1,376,011 using techniques in the paper

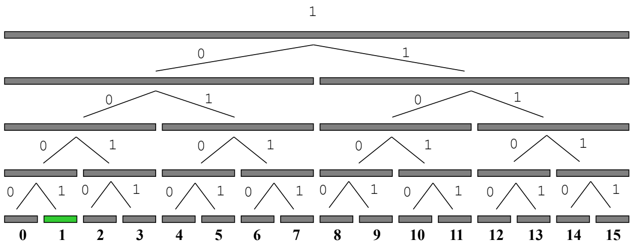
Segment Columns

- Try to avoid range query on index
- Keep columns for normal timestamp
- Add columns for segments
  - Note at most two segments of any given length
  - Column s2 – first segment of length 2
  - Column s2x – other segment of length 2
  - Nulls are common in segment columns

Data	*Time Metadata*									
Dept	**Start**	**Stop**	**s1**	**s2**	**s4**	**s8**	**s1x**	**s2x**	**s4x**	**s8x**
...	1	11	10001	1001	101				110	
...	2	3		1001						
...	5	6	10101				10110			

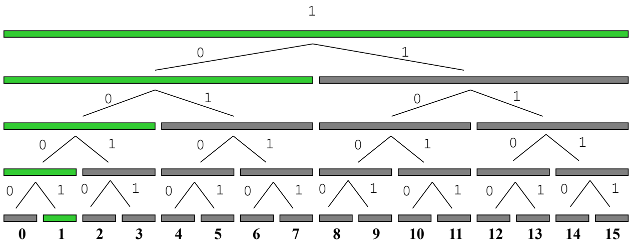
Segment Endpoint Containment

- Precompute and store segments that contain a start or stop time
- Consider the interval [1, 11]



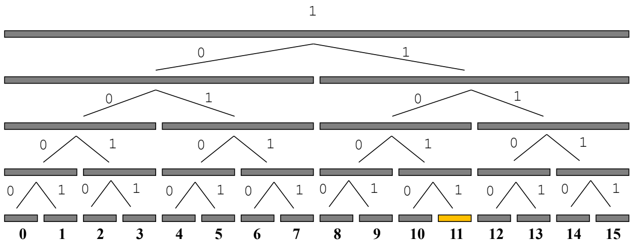
Segment Endpoint Containment

- Precompute and store segments that contain a start or stop time
- Consider the interval [1, 11]



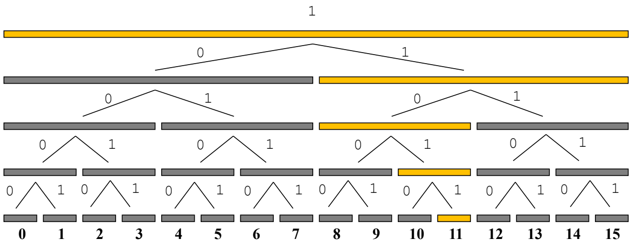
Segment Endpoint Containment

- Precompute and store segments that contain a start or stop time
- Consider the interval [1, 11]



Segment Endpoint Containment

- Precompute and store segments that contain a start or stop time
- Consider the interval [1, 11]



Help Determine Endpoint Containment

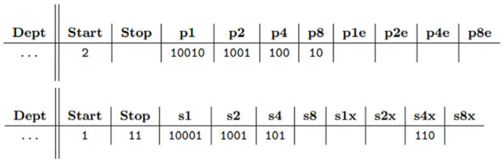
Add columns for segments that could contain the start and stop points

- Prefix column p2 – What segment of length 2 contains start?
- Prefix column p2e – What segment of length 2 contains stop?

Data	*Time Metadata*									
Dept	**Start**	**Stop**	**p1**	**p2**	**p4**	**p8**	**p1e**	**p2e**	**p4e**	**p8e**
...	1	11	10001	1000	100	10	11011	1101	110	11
...	2	3	10010	1001	100	10	10011	1001	100	10
...	5	6	10101	1010	101	10	10110	1011	101	10

Using

- Is 2 contained in [1,11]?

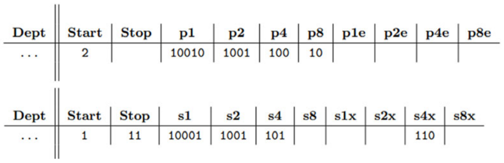


Dept	Start	Stop	p1	p2	p4	p8	p1e	p2e	p4e	p8e
...	2		10010	1001	100	10				

Dept	Start	Stop	s1	s2	s4	s8	s1x	s2x	s4x	s8x
...	1	11	10001	1001	101				110	

Using

- Is 2 contained in [1,11]?



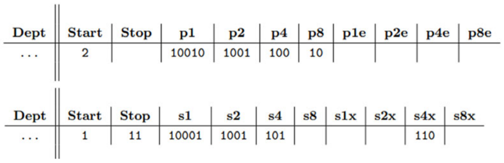
Dept	Start	Stop	p1	p2	p4	p8	p1e	p2e	p4e	p8e
...	2		10010	1001	100	10				

Dept	Start	Stop	s1	s2	s4	s8	s1x	s2x	s4x	s8x
...	1	11	10001	1001	101				110	

- Yes, p2 == s2

Using

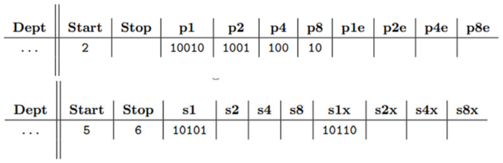
- Is 2 contained in [1,11]?



Dept	Start	Stop	p1	p2	p4	p8	p1e	p2e	p4e	p8e
...	2		10010	1001	100	10				

Dept	Start	Stop	s1	s2	s4	s8	s1x	s2x	s4x	s8x
...	1	11	10001	1001	101				110	

- Yes, p2 == s2
- Is 2 contained in [5,6]?

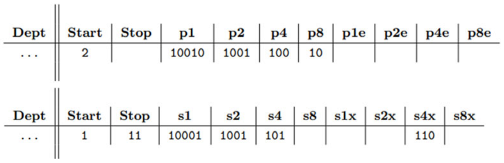


Dept	Start	Stop	p1	p2	p4	p8	p1e	p2e	p4e	p8e
...	2		10010	1001	100	10				

Dept	Start	Stop	s1	s2	s4	s8	s1x	s2x	s4x	s8x
...	5	6	10101				10110			

Using

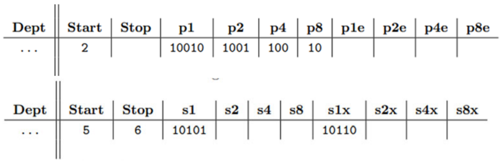
- Is 2 contained in [1,11]?



Dept	Start	Stop	p1	p2	p4	p8	p1e	p2e	p4e	p8e
...	2		10010	1001	100	10				

Dept	Start	Stop	s1	s2	s4	s8	s1x	s2x	s4x	s8x
...	1	11	10001	1001	101				110	

- Yes, p2 == s2
- Is 2 contained in [5,6]?



Dept	Start	Stop	p1	p2	p4	p8	p1e	p2e	p4e	p8e
...	2		10010	1001	100	10				

Dept	Start	Stop	s1	s2	s4	s8	s1x	s2x	s4x	s8x
...	5	6	10101				10110			

- No, no *N* such that s*N* == p*N* or s*N*x == p*N*

Outline

- Timestamp representation
- Application to nonsequenced queries
- Evaluation
- Conclusion

Experiment Setup

- Test machine
  - Oracle Cloud Instance
  - 4 CPUs – 2.4 GHZ
  - 32GB RAM
  - 1 TB SSD drive
  - Linux
- Test DBMS
  - Postgres, version 14
  - Made no adjustments to out-of-the-box settings
  - EXPLAIN – optimizes and generates query execution plan
  - EXPLAIN ANALYZE – runs query as well

Evaluation

- Compare timestamped vs. log-segmented
- One relation
  - Timestamped

```
Employees(id, name, department, start, stop)
```

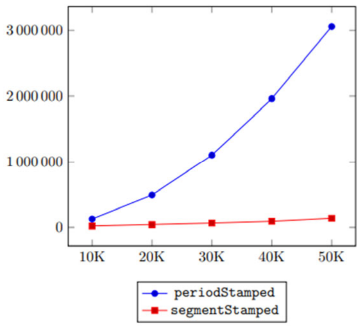
  - Log-segmented

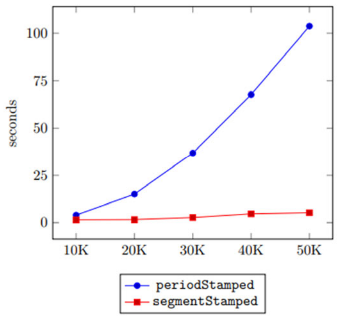
```
Employees(id, name, department, start, stop,
          s1, s2, ..., s19, s1x, s2x, ..., s19x,
          p1, p2, ..., p19, p1e, p2e, ..., p19e)
```

Experiment Data and Queries

- Test data
  - Synthetically generated
  - 100 departments, 90% different names
  - 10K to 50K tuples
  - Timeline of 2^19
  - Timestamps 2^8, randomly generated
- **Create indexes for everything!**
- Test query
  - Join of employee with itself, only on the temporal attributes
    Focus on timestamps, not non-temporal columns
  - Three predicates for join
    - Overlaps
    - Contains
    - Starts

Evaluation - Overlaps





Overlaps Query Execution Plan

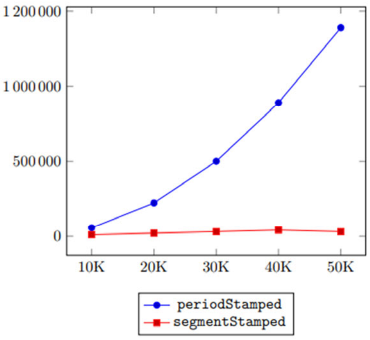
- SQL query WHERE clause **is ugly**

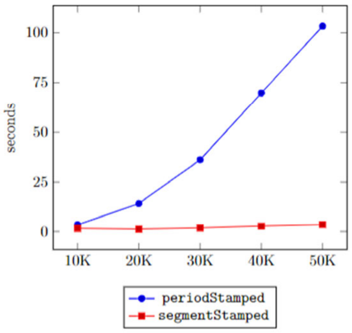
```
WHERE ...
    (x.s1 = y.p1 OR r.s2 = y.p2 OR ... OR x.s19 = y.p19
  OR x.s1 = y.p1e OR r.s2 = y.p2e OR ... OR x.s19 = y.p19e
  OR y.s1 = x.p1 OR y.s2 = x.p2 OR ... OR y.s19 = x.p19
  OR y.s1 = x.p1e OR y.s2 = x.p2e OR .. OR y.s19 = x.p19e)
```

- Query execution plan

```
Nested Loop  (cost=12.13..102716.47 rows=120094 width=20)
  ->  Seq Scan on empt r  (cost=0.00..417.00 rows=10000 width=176)
  ->  Bitmap Heap Scan on empt s  (cost=12.13..18.11 rows=18 width=84)
  Recheck Cond: ((s1 = r.p1) OR (s2 = r.p2) ... OR (s262144 = r.p262144x) OR (s524288 = r.p524288x))
    ->  BitmapOr  (cost=12.13..12.13 rows=18 width=0)
      ->  Bitmap Index Scan on foos1  (cost=0.00..0.30 rows=1 width=0)
                                 Index Cond: (s1 = r.p1)
      ->  Bitmap Index Scan on foos2  (cost=0.00..0.30 rows=1 width=0)
                                 Index Cond: (s2 = r.p2)
      ->  Bitmap Index Scan on foos4  (cost=0.00..0.30 rows=1 width=0)
      ...
      ->  Bitmap Index Scan on foos524288x  (cost=0.00..0.30 rows=1 width=0)
                                 Index Cond: (s524288 = s_1.p524288)
```

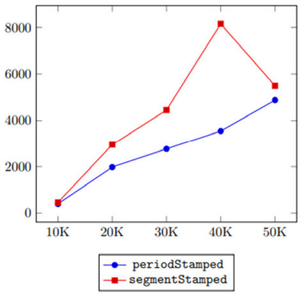
Evaluation - Contains

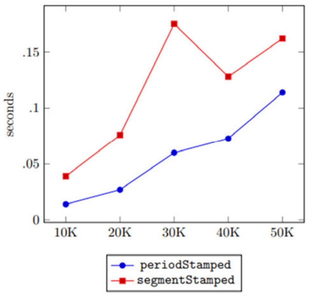




Disadvantages

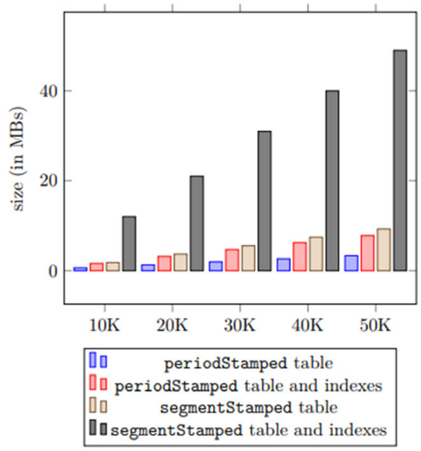
- Starts performs worse with log-segmented





- Space cost increases (next slide)

Results – DB Size

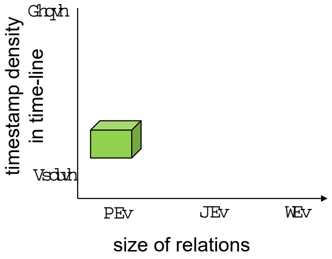


Outline

- Timestamp representation
- Application to nonsequenced queries
- Evaluation
- Conclusion

Conclusion

- Query optimization technique
  - Log segmented stores both normal and log-segmented timestamps
  - Run optimizer on both, choose best plan
  - Downside is extra space
  - Additional benefit – sequenced semantics!
- Tested only a small part of query optimization space



- Size of result, time-line size, value conditions, etc.

Future Work

- Log-segmented Cypher
- New temporal hash-join technique