

Temporal Coalescing

Roger Schneider

17.4.2010

Content

Introduction

Example

Definition of Coalescing

Implementation of the Coalescing Operator

Implementation in pure SQL

OLAP Implementation

Performance of Coalescing

Conclusion

Motivating Example

AHVNr	PNo	WCode	valid time (VT)
1000	10	a	08:00, 10:00
1000	10	c	10:00, 12:00
1000	11	b	12:00, 17:00

Table: Relation works_on

Temporal Query:

Which time period worked employee 1000 the day?

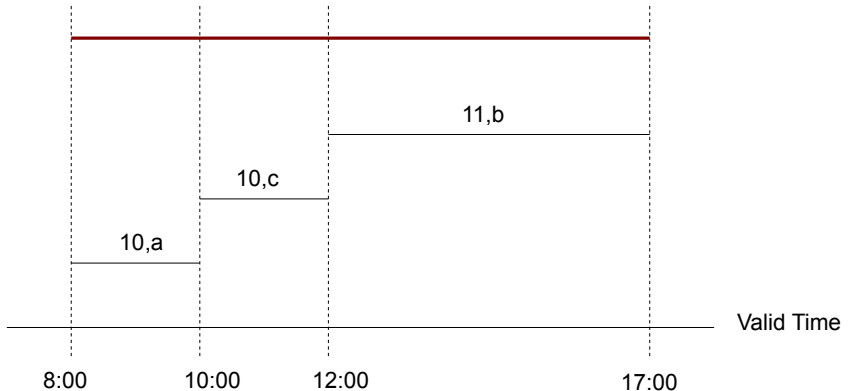


Figure: Time periods on which employee 1000 worked a day

AHVNr	PNo	WCode	valid time (VT)
1000	10	a	08:00, 10:00
1000	10	c	10:00, 12:00
1000	11	b	12:00, 17:00

Table: Uncoalesced relation works_on

AHVNr	VT
1000	08:00, 17:00

Table: Coalesced relation works_on

- ▶ Tuples reduction (from 3 -> to 1)
- ▶ Lost of information (PNo, WCode, time periods)

Intuitive Proceeding

1. search for value equivalent tuples on the explicit attribute
2. order the tuples by time period
3. build the maximum period
4. delete tuples which time periods contained in the maximum period

The importance of the coalescing operation

- ▶ reducing tuples
- ▶ new queries on time periods
- ▶ avoiding incorrect answers in temporal databases
 - ▶ example

Example

Name	VT
Joe	2010-01-01, 2010-01-15
Joe	2010-01-14, 2010-01-31

Table: Uncoalesced relation

Name	VT
Joe	2010-01-01, 2010-01-31

Table: coalesced

- ▶ Was Joe employed on January the 12th?
both relation have the same result
- ▶ Worked Joe the entire month of January?
the result is negative on the uncoalesced relation

Auxiliary Predicates

1. we have to test the explicit attributes values of two tuples on equality.

$$value_eq(x, y) := (x[A] = y[A])$$

2. the next test checks the time-periods of the two tuples on adjacency

$$adj([S_1 - E_1], [S_2 - E_2]) := (E_1 = S_2 \vee S_1 = E_2))$$

3. the last test verifies, if the two tuples are overlapping

$$ovlp([S_1 - E_1], [S_2 - E_2]) := (\exists c((S_1 \leq c < E_1) \wedge (S_2 \leq c < E_2)))$$

Let $R = (A_1, \dots, A_N \parallel VT)$ be a valid-time relation schema. A instance r of R is coalesced if and only if

$$\begin{aligned}
 is_coal(r) := & \forall x, y \in r (x = y \\
 & \vee \neg value_eq(x, y) \\
 & \vee (\neg ovlp(x[VT], y[VT]) \wedge \neg adj(x[VT], y[VT])))
 \end{aligned}$$

Implementation of the Coalescing Operator

- ▶ Implementation with standard SQL or embedded SQL
- ▶ Implementation in the main memory
- ▶ Implementation in the DBMS
- ▶ Reorganization of the schema

Non-iterative Implementation

Which problems exist?

- ▶ SQL standard doesn't include an order on tuples
- ▶ SQL standard doesn't contain functions which take the first or last tuple in a selection



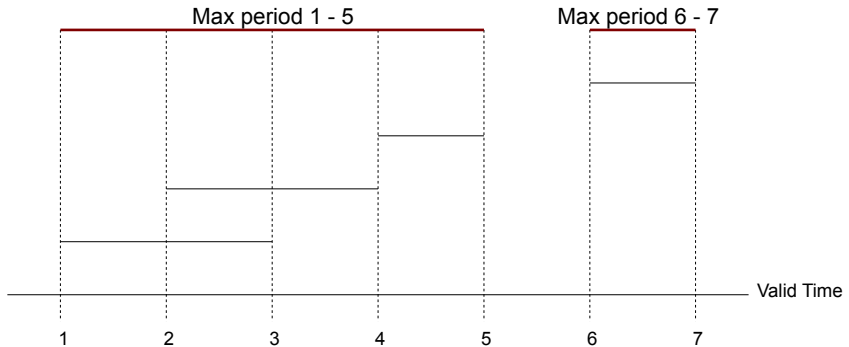
- ▶ search for two value-equivalent tuples, (**F**)irst and (**L**)ast
- ▶ ensure that there are no holes between F.Start and L.End
- ▶ ensure that only maximum periods result

Example

c	S(tart)	E(nd)
c	1	3
c	2	4
c	4	5
c	6	7

Table: Uncoalesced relation r

Timeline



First and Last Period

```

SELECT DISTINCT f.S, l.E, f.c
FROM r AS f, r AS l
WHERE f.S < l.E
        AND f.c = l.c
  
```

c	S(tart)	E(nd)
c	1	3
c	1	4
c	1	5
c	1	7
c	2	3
c	2	4
c	2	5
c	2	7
c	4	5
c	4	7
c	6	7

Periods without Holes

```
AND NOT EXISTS ( SELECT *  
                  FROM r AS m  
                  WHERE m.c = f.c AND f.S < m.S AND m.S < l.E  
                  AND NOT EXISTS ( SELECT *  
                                   FROM r as a1  
                                   WHERE a1.c = f.c  
                                   AND a1.S < m.S AND m.S <= a1.E ) )
```

c	S(tart)	E(nd)
c	1	3
c	1	4
c	1	5
c	2	3
c	2	4
c	2	5
c	4	5
c	6	7

Possibles Maximum Periods

```
AND NOT EXISTS ( SELECT *  
                  FROM r AS a2  
                  WHERE a2.c = f.c  
                        AND ( a2.S < f.S AND f.S <= a2.E  
                              OR a2.S <= l.E AND l.E < a2.E ) )
```

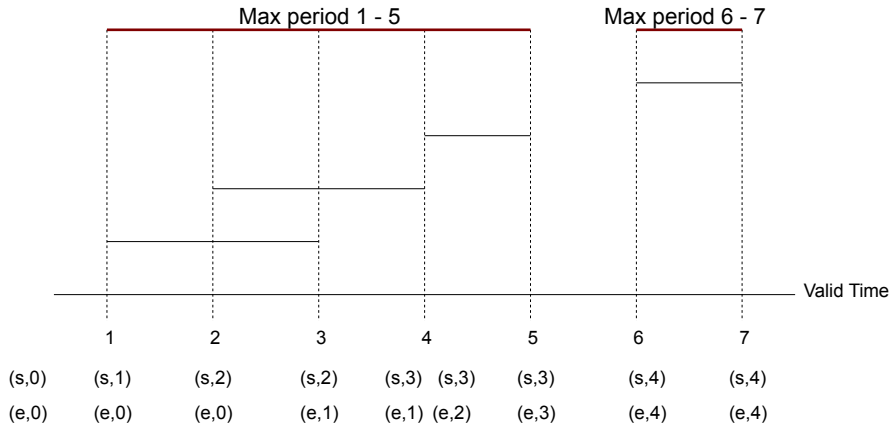
c	S(tart)	E(nd)
c	1	5
c	1	7
c	6	7

SQL Code

```
SELECT DISTINCT f.S, l.E, f.c
FROM r AS f, r AS l
WHERE f.S < l.E
      AND f.c = l.c
      AND NOT EXISTS ( SELECT *
                        FROM r AS m
                        WHERE m.c = f.c AND f.S < m.S AND m.S < l.E
                        AND NOT EXISTS ( SELECT *
                                        FROM r as a1
                                        WHERE a1.c = f.c
                                        AND a1.S < m.S AND m.S <= a1.E ) )

AND NOT EXISTS ( SELECT *
                  FROM r AS a2
                  WHERE a2.c = f.c
                        AND ( a2.S < f.S AND f.S <= a2.E
                            OR a2.S <= l.E AND l.E < a2.E ) )
```

Single Scan Coalescing Algorithm



Possibles Time Periods

c	ts	Start_ts	End_ts	Cur_Total_ts	Prev_Total_ts
c	1	1	0	1	0
c	2	1	0	2	1
c	3	0	1	1	2
c	4	1	0	2	1
c	4	0	1	1	2
c	5	0	1	0	1
c	6	1	0	1	0
c	7	0	1	0	1

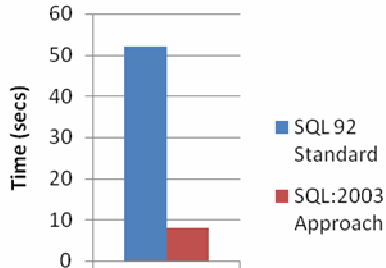


Figure: Database with more 93MB

Conclusion

- ▶ The Temporal Coalescing Operator is an important operator in temporal databases
(Reduction of tuples, new queries)
- ▶ Coalescing can implemented in SQL *(No extension for SQL is required)*
- ▶ Temporal Coalescing is complex to implement
(In SQL:1992 several nested "not exists"-clauses)
- ▶ Temporal Coalescing implemented in SQL is time consuming
(complex)