

Temporal Colaescing

Roger Schneider

Matrikelnummer: 89-708-903

Email: roschneider@bluewin.ch

May 25, 2010

supervised by Prof. Dr. M. Böhlen, Prof. Dr. D. Kossmann and Prof. Dr. P. Widmayer



University of Zurich
Department of Informatics



Contents

1	Introduction	3
1.1	Introductory example	3
1.2	Importance of Coalescing	4
2	Definitions	5
2.1	Summary	5
3	Implementation of the Coalescing Operator	6
3.1	Implementation with non-iterative SQL	7
3.2	Implementation with OLAP-features	9
3.3	Performance of the implementations	11
4	Conclusion	11

1 Introduction

This report is based on the paper [BSS96] and outlines some aspects of the coalescing operator.

Coalescing operation merges value-equivalent tuples with same non-timestamp attributes and adjacent or overlapping time intervals.

1.1 Introductory example

In the example we have the following relation (works_on):

First attribute is the social security number (AHVNr), the second is the project number (PNo), the third is the working code (WCode) and the fourth attribute is the valid time of the tuple with start and end time.

So, a person (AHVNr) works on a project (PNo) with a working code (WCode) over a timestamped period (Valid Time with start and end time).

For the personnel office it could be important to know how long an employee worked the day. For this reason we have to identify the maximum time period of each employee.

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 1: Relation works_on

A possible temporal Query:

Which time interval worked employee 1000 the day?

In the timeline diagram figure 1 we can see the time periods employee 1000 worked the day. There are three adjacent time intervals which add up to the maximum time period of the day.

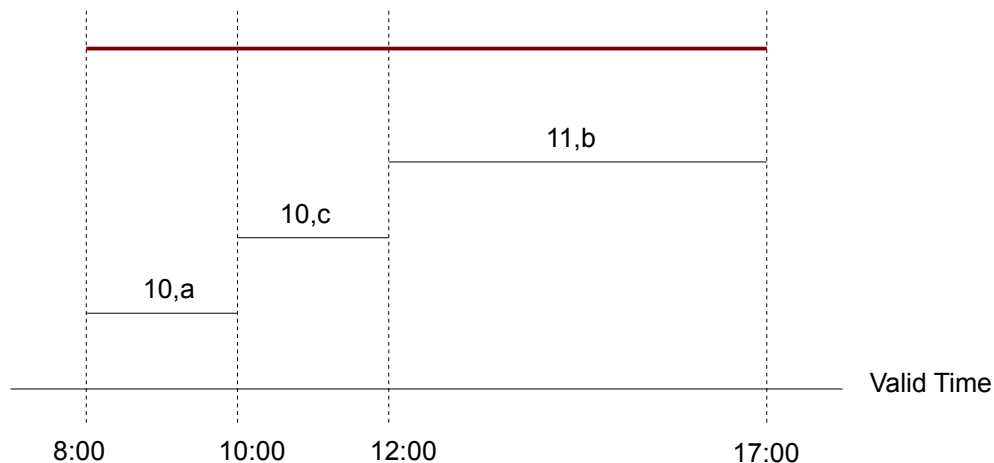


Figure 1: Time periods on which employee 1000 worked a day

The result of the query is one tuple in the result relation. This tuple represents the maximal time period.

AHVNr	VT
1000	08:00, 17:00

Table 2: Coalesced relation works_on with one maximum period

The result relation shows an important fact in coalescing. We have loss the information on the project number and the working code. So, it is impossible to rebuild the start relation with all the tuples from the coalesced relation. There is always loss of information.

But the result shows too, we have a reduction from 3 tuples to 1 tuples in the result relation. We have to consider the periods to construct a coalescing operator. The condition for the periods is, that the time periods have to be adjacent. In such a case we have achieved the condition to coalesce the tuple, if the explicit attribute values are equivalent additional.

1.2 Importance of Coalescing

The previous example demonstrates the obvious behaviour of the coalescing operator. With the coalescing operator in a temporal database we could execute the following operations:

- Reducing tuples in a temporal relation:
with reduced relations we can compute in shorter time
- New queries over time-periods:
in a temporal database it is possible to pose new queries which could not posed in standard databases. These are queries on time intervals mainly.
- Avoiding incorrect answers in temporal databases:
the same queries on uncoalesced and coalesced relations can produce different results. See the following explicative example

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

Table 3: Uncoalesced relation

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

Table 4: coalesced

The following queries induce different results:

- Was Joe employed on January the 12th?
the valid time and the tuple-by-tuple evaluation produce the same result in the representations in table 3 and table 4

- Worked Joe the entire month of January?
the results in table 3 is empty, because no tuple could found in the relation. The result in table 4 is correct

The first query deals with single timestamp while the second query does with time interval. We can say that the relations have different semantics which are not compatible to the coalescing operation.

2 Definitions

In the first example we have seen, that we have to find tuples which are value-equivalent on the explicit attribute and have adjacent time-periods. Additionally coalescing has to deal with overlapping time periods, too.

First we have to define the timestamps of the tuples. [BSS96] chose in their report for the data model a first normal model with open periods $([t_s - t_e))$ timestamps. But other temporal data models can be constructed similarly. We define the relation schema $R = (A_1, \dots, A_n || VT)$. To simplify the definition of coalescing, we define three auxiliary predicates.

1. we have to test the explicit attribute value of two tuples on equality

$$value_eq(x, y) := (x[A] = y[A]) \quad (1)$$

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

$$adj([S_1 - E_1), [S_2 - E_2)) := (E_1 = S_2 \vee S_1 = E_2)) \quad (2)$$

3. the last test verifies if two tuples are overlapping

$$ovlp([S_1 - E_1), [S_2 - E_2)) := (\exists c((S_1 \leq c < E_1) \wedge (S_2 \leq c < E_2))) \quad (3)$$

Now, we can verifying a relation if the relation is coalesced or not. Let r be an instance of a relation R :

$$\begin{aligned} is_coal(r) := & \forall x, y \in r (x = y \\ & \vee \neg value_eq(x, y) \\ & \vee (\neg ovlp(x[VT], y[VT]) \vee \neg adj(x[VT], y[VT]))) \end{aligned} \quad (4)$$

A relation r is coalesced, if all pair of distinct tuples from the relation are either not value-equivalent, or, if they are value-equivalent, then they must be non-adjacent and non-overlapping.

2.1 Summary

We demonstrated some properties of the coalescing operator. Before we will implement this operator, we should point out the important qualities:

- the operator can expand for more than one explicit attribute
- the coalescing operation is not reversible. We lose information always.
- to build the maximum time periods and then to delete the non-maximal valid-times has similarity to *duplicate elimination* in conventional databases. This operation is very time consuming and complex [BD83]. So the coalescing operation has to be time consuming, too.

3 Implementation of the Coalescing Operator

There exist some possibilities to implement the coalescing operator.

- **Implementation with standard SQL or embedded SQL**
 - advantages:
implementation by the user and no condition by standard SQL
 - disadvantages:
less efficient performance
- **Implementation in the main memory**
 - advantages:
possibilities to implement sophisticated algorithms for good performance
 - disadvantages:
reimplementation of DBMS operations and high entry-costs
- **Implementation in the DBMS**
 - advantages:
the time consuming operations could implemented directly in the DBMS
 - disadvantages:
there are some adaptations at the implemented SQL for temporal operations
- **Reorganization of the schema**
 - advantages:
only standard SQL
 - disadvantages:
existing schemas have to be reengineered

We decided to appoint the implementations in standard SQL. This part appears interesting for us, while the user can implement the coalescing operator by himself and there are no conditions at the DBMS, exception it has to support standard SQL.

3.1 Implementation with non-iterative SQL

To implement the coalescing operator there exist problems in the definition of SQL. SQL standard doesn't include an order on tuples and doesn't contain functions which can take the first or last tuple in a selection. Two deficits which the implementation of coalescing in pure SQL make tricky. [BSS97] demonstrate that it is possible to implement coalescing with the deficits on standard SQL. An implementation in non-iterative SQL has the advantage, that the coalescing operator can implemented in older versions of SQL (e.g. SQL:1992) without procedural extension and in standard relational databases.

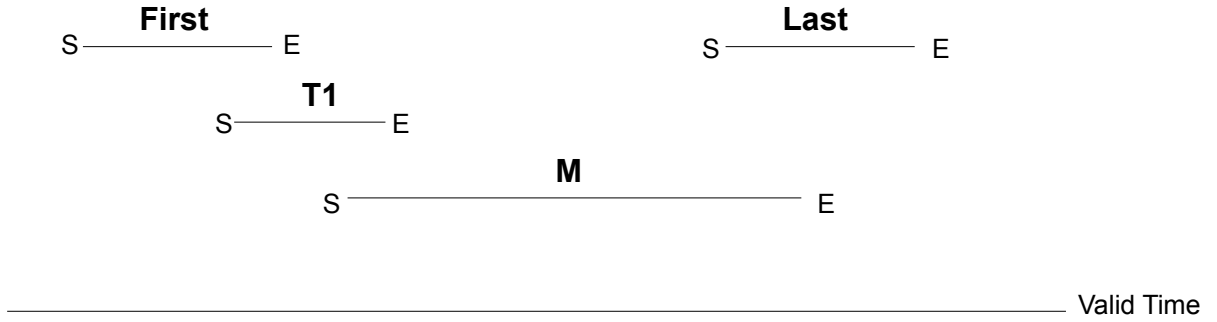


Figure 2: Non-iterative Coalescing

1. search for two value-equivalent tuples, which could possibly be the same. Determine the First and the Last period. ($First.S$ and $Last.E$)
2. ensure that there are no holes between $First.E$ and $Last.S$. All start points $M.S$ of value-equivalent tuples M are extended towards $First.S$ by another value-equivalent tuple $T1$. $T1$ could be $First$ or $First$ overlaps $Last$. M need not overlap $Last$, so there must exist another M .
3. ensure that only maximal periods result. First and Last cannot be part of a larger value-equivalent tuple $T2$.

With this model we can arrange the first-order logic formula:

$$\begin{aligned}
 & r(X \parallel [S - _]) \wedge r(X \parallel [_ - E]) \wedge S < E \wedge \\
 & \forall A (r(X \parallel [A - _]) \wedge S < A < E \Rightarrow \\
 & \exists U, V (r(X \parallel [U - V]) \wedge U < A \leq V) \wedge \\
 & \neg \exists A, B (r(X \parallel [A - B]) \wedge (A < S \leq B \vee A \leq E < B))
 \end{aligned} \tag{5}$$

The first-order logic formula has to be translated to SQL. For this purpose we have to eliminate the $\forall$ and implications. After that, there are several nested *NOT EXISTS* clauses and self-joins in the SQL code [BSS97].

```

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 t1
            WHERE t1.c = f.c
            AND t1.S < m.S AND m.S <= t1.E))

    AND NOT EXISTS (SELECT *
        FROM r AS t2
        WHERE t2.c = f.c
        AND (t2.S < f.S AND f.S <= t2.E
        OR t2.S <= l.E AND l.E < t2.E))

```

We will demonstrate the non-iterative coalescing operator by an example.
 Example: Let be the following relation with 4 tuples.

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

Table 5: Example relation

The example contains time-periods which are adjacent, overlapping and there exists a hole.
 Figure 3 shows the timeline diagram of the relation.

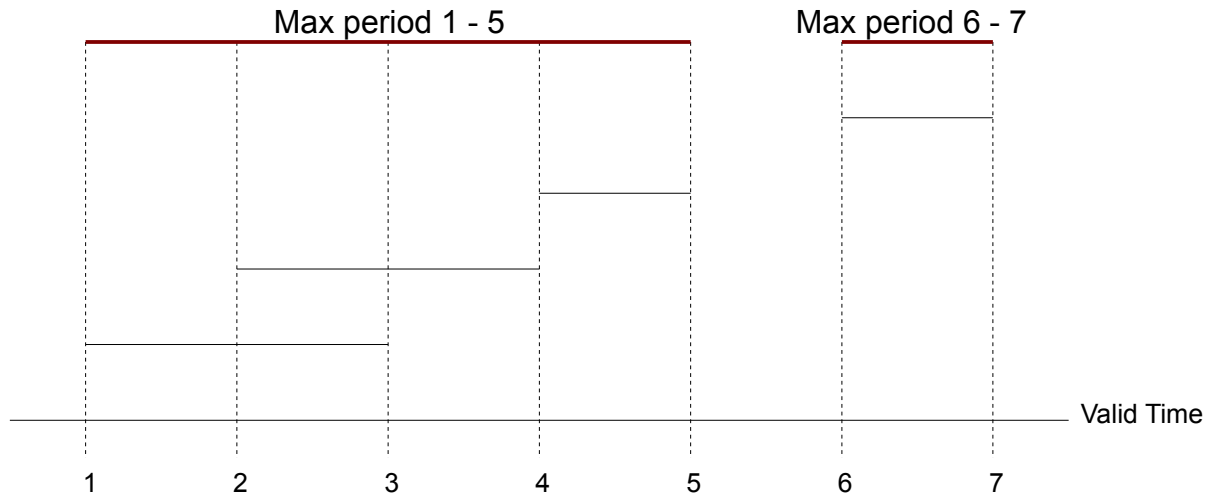


Figure 3: Timeline of the example tuples

In the first step, with the self-join table 6, we get all possible time intervals which fulfil the condition $First.S < Last.E$

In the first *NOT EXISTS* table 7, we get all possible time intervals which are adjacent or overlapping, but no holes.

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

Table 6: Result relation after self-join

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

Table 7: Relation without holes between time-periods

The next *NOT EXISTS* and the join, we get all possible maximum time-intervals, table 8. The last *NOT EXISTS* chose the maximum time-intervals, table 9.

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

Table 8: Possibles maximum periods

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

Table 9: All maximum periods

The algorithm has only executed once.

After execution the algorithm, it have to execute the deletion operation. The deletion operation eliminates the time intervals which are inclose in the maximal time periods.

3.2 Implementation with OLAP-features

The OLAP (online analytical processing) functionality extends the SQL. The execution is in the main memory. We can suppose, that the elapsed time for coalescing should be less than in SQL:1992 [ZWZ08]. These new extensions allow to implement the coalescing operator too in SQL.

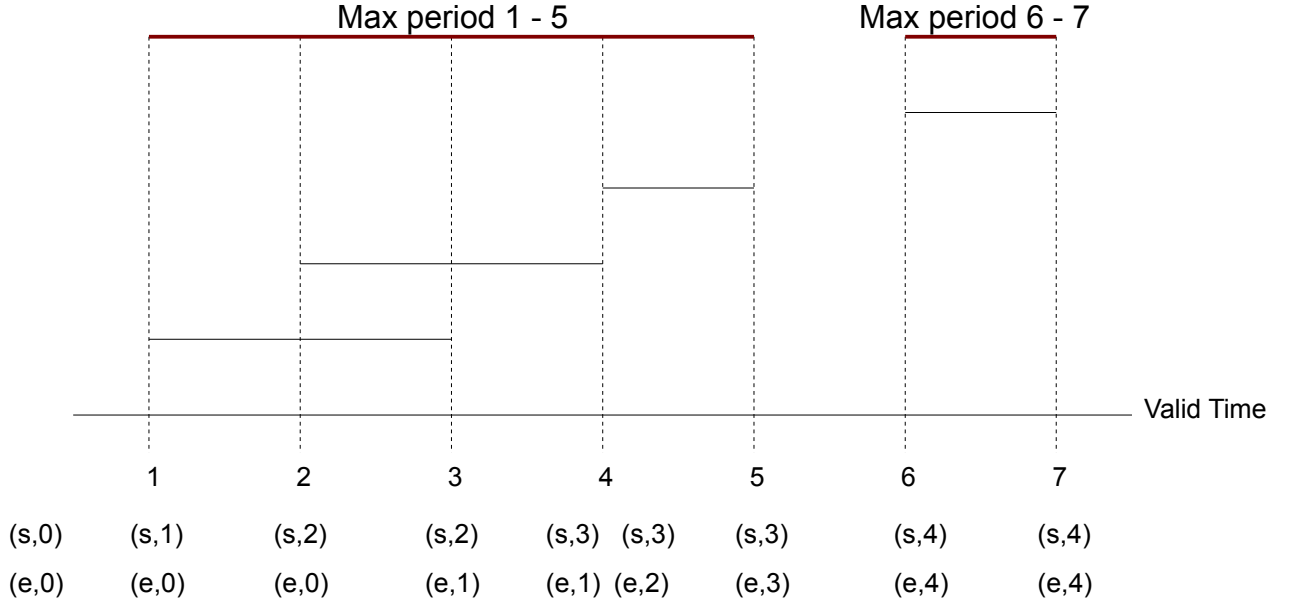


Figure 4: Coalescing Algorithm with OLAP

For each timestamp we need the information if the timestamp is start oder end of the time-period. So, we introduce two new variables which were updated for every new timestamp. Output all coalesced timestamps t_i to t_j where t_{i-1} has $(s, m)/(e, m)$ and t_j has $(e, n)/(e, n)$. Intuitively, at timestamp t_i , all previous periods have been output as one coalesced period, and a new coalescable period begins from t_i . At t_j we have seen equal number of Start_ts and End_ts, which means a coalesced period ends at t_j timestamp, and t_j should be End_ts value of our coalesced output.

We use the same example like non-iterative SQL implementation.

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

Table 10: Intermediate result

In a temporary table $T1$ were collected all S and E timestamps from the input tuples in Start_ts and End_ts. A table $T2$ keeps the difference between the total count of Start_ts and the total count of End_ts timestamps until t_i (stored as Cur_Total_ts). Similary, it keeps the difference between the total count of Start_ts and the total count of End_ts timestamps until t_{i-1} (stored as Prev_Total_ts). Thus if Cur_Total_ts is 0, it means that there are equal number of Start_ts

and End_ts timestamps before t_i , so t_i should be a end timestamp for one of the coalesced periods. If Prev_Total_ts is 0, it means t_i should be a start timestamp for one of the coalesced periods.

The code is complex, but we have to consider that the operations are executed in the main memory.

3.3 Performance of the implementations

[UP09] implemented the non-iterative coalescing with SQL:92 and the OLAP-features coalescing with SQL:2003. They created a database with around 93MB and tested the both implementation against.

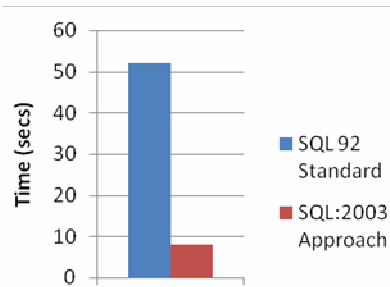


Figure 5: Performance of both SQL implementations. Database with more then 93MB

The comparison demonstrates the efficiency of the SQL:2003 implementation. With the new performance it is possible to implement the coalescing operation in conventional database.

4 Conclusion

The temporal coalescing operation is very important in temporal databases.

The same query on an uncoalesced and a coalesced relation can produce different results.

A coalesced relation has lose information. So coalescing is not a reversible operation and coalescing restructures the relation.

A relation r is coalesced, if all pair of distinct tuples from the relation are either not value-equivalent, or, if they are value-equivalent, then they must be non-adjacent and non-overlapping. Coalescing has some similarity with the duplicate elimination which is very time consuming. So, coalescing is a time consuming and complex operation, too.

We demonstrated that coalescing can implemented in SQL:1992 without procedural extension or recursive constructs.

An augmentation of performance could discovered in the SQL:2003 implementation and the OLAP-features.

These results demonstrate the problems of the coalescing operator. The interplay of coalescing and other temporal operators rests unknown.

References

- [BD83] Dina Bitton and David J. DeWitt. Duplicate record elimination in large data files. *ACM Trans. Database Syst.*, 8(2):255–265, 1983.
- [BSS96] Michael H. Böhlen, Richard Thomas Snodgrass, and Michael D. Soo. Coalescing in temporal databases. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 180–191, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.
- [BSS97] Michael H. Böhlen, Richard Thomas Snodgrass, and Michael D. Soo. Coalescing in temporal databases. Technical report, TimeCenter TR-9, 1997.
- [UP09] K. Unnikrishnan and K. V Pramod. On implementing temporal coalescing in temporal databases implemented on top of relational database systems. In *ICAC3 '09: Proceedings of the International Conference on Advances in Computing, Communication and Control*, pages 153–156, New York, NY, USA, 2009. ACM.
- [ZWZ08] Xin Zhou, Fusheng Wang, and Carlo Zaniolo. Efficient temporal coalescing query support in relational database systems. 2008.