

Temporal Statement Modifiers

David Caspar

May 30, 2010

Contents

1	Introduction	1
1.1	Temporal data	1
2	Desiderata for a temporal extension	1
2.1	Upward compatibility	2
2.2	Temporal upward compatibility	2
2.3	S -reducibility	3
2.4	Extended S -reducibility	4
2.5	Interval preservation	4
2.6	Non-restrictiveness	5
2.7	Summary	6
3	Extending the syntax of SQL-based languages	6
4	Defining the formal semantics	7
4.1	The temporal relational algebra	7
4.2	Translating temporal statements to algebra expressions	9
5	Conclusion	11

1 Introduction

This report discusses an approach how to extend a non-temporal query language with temporal language features in such a way that the extended language provides full support for temporal queries while maintaining support for non-temporal queries as well. It is an outline of a paper by Michael H. Böhlen, Christian S. Jensen and Richard T. Snodgrass, entitled “Temporal Statement Modifiers” and published in 2000 (for details, see [2]).

The report is divided into three parts. First, a set of desiderata is presented that a temporal extension of a non-temporal query language has to fulfill. Then, based on the desiderata, a temporal extension for SQL-92 is presented to illustrate how a given query language can be extended syntactically. Finally, the precise semantics of the extended language are presented, introducing some relational algebra operators working on temporal relations.

1.1 Temporal data

When talking about data varying over time, two temporal aspects are of special interest. The first one is *valid time*, that is the times when the fact was or will be valid in the modeled reality ([2], page 2). Secondly, the *transaction time* of a fact is the times when the fact has been stored as current in the database. Since all database facts have a valid time and a transaction time, the report adopts the term *temporal* from the paper when at least one of valid or transaction time is captured. Relations that store valid time are called *valid-time relations*, whereas those relations storing transaction time or both kind of temporal information are called *transaction-time relations* and *bi-temporal relations*, respectively. In contrast, a *non-temporal* relation stores no temporal data of the mentioned form.

2 Desiderata for a temporal extension

Throughout this section, a set of desiderata is examined that a temporal extension of a query language has to fulfill.

2.1 Upward compatibility

Upward compatibility ensures that existing non-temporal databases and query statements are still valid in the new temporal data model, that is they produce the same results in the new temporal data model as they did before in a non-temporal environment.

Example 1. Consider a non-temporal relation **Employee** (where the attribute Valid Time refers to one of the possibilities for capturing a time period in SQL, e.g. in the form of two timestamps)

Name	Salary	Valid Time
Paul	20000	1999-2002
Sarah	30000	1999-2004
Peter	20000	2001-2002
Peter	25000	2003-2006
Lisa	25000	2007-2009

and a non-temporal query `SELECT COUNT(*) AS "Employees" FROM Employee`. The result of this query in a temporal data model should be the same as in a non-temporal model, namely

Employees
5

The formal definition of upward compatibility is as follows ([2], page 5):

Definition 2. Let $M^x = (DS^x, QL^x)$ and $M^y = (DS^y, QL^y)$ be two data models¹. Model M^y is upward compatible with model M^x iff

$$\forall db^x \in DS^x : db^x \in DS^y \text{ and } \forall s^x \in QL^x : s^x \in QL^y$$

(*syntactic upward compatibility*) and

$$\forall db^x \in DS^x, \forall s^x \in QL^x : [s^x(db^x)]_{M^x} = [s^x(db^x)]_{M^y}$$

where $[s(db)]_M$ denotes the result of applying a statement s to a database db in a data model M .

2.2 Temporal upward compatibility

Temporal upward compatibility is an expansion of upward compatibility, ensuring that legacy non-temporal queries continue to work when applied to temporal relations. The formal definition is as follows ([2], page 7):

Definition 3. Let a temporal and a non-temporal data model be given by $M^t = (DS^t, QL^t)$ and $M^s = (DS^s, QL^s)$. Let τ be an operator that changes the type of a relation from non-temporal to temporal. Let $U = u_1, u_2, \dots, u_n$ denote a sequence of update operations. Model M^t is temporal upward compatible with model M^s iff M^t is upward compatible with M^s and

$$\forall db^s \in DS^s, \forall U, \forall q^s \in QL^s : [q^s(U(db^s))]_{M^s} = [q^s(U(\tau(db^s)))]_{M^t}$$

It follows from temporal upward compatibility that a non-temporal query is evaluated on the latest snapshot² of a temporal relation.

Example 4. Suppose the relation **Employee** from the previous example was transformed into a temporal relation **EmployeeTemp**³, storing the same information, but this time with automatically captured temporal information for the valid time of a record (denoted as VTIME):

Name	Salary	VTIME
Paul	20000	1999-2002
Sarah	30000	1999-2004
Peter	20000	2001-2002
Peter	25000	2003-2006
Lisa	25000	2007-2009

¹By convention, a data model $M = (DS, QL)$ consists of a set of data structures DS , containing all databases, schemes and associated instances expressible by M , and a language QL for querying these data structures, which contains all update and query statements applicable to some database in DS .

²A snapshot of a temporal relation at a given time point t is a non-temporal relation, containing all tuples that are valid at time point t .

³The transformation can be done by using non-sequenced statements presented later in this report.

If one had evaluated the non-temporal query `SELECT COUNT(*) AS "Employees" FROM EmployeeTemp` in the year 2004, the result of this query would have been

Employees
2

since only Sarah's and Peter's records are valid during the year 2004.

2.3 S -reducibility

A temporal data model is an S -reducible extension of a non-temporal data model iff for every non-temporal query q , there exists a corresponding temporal query q^T such that if one takes the (temporal) result of q^T evaluated on a temporal relation r , the result's snapshot at a time point t is the same as if q had been evaluated on the snapshot of r at time t . Furthermore, q^T must look syntactically similar to q .

For a formal definition, we need to introduce a new algebraic operator, the bi-temporal timeslice operator $\tau_{(c^{tt}, c^{vt})}^{M^{bi}, M}$, which takes a bi-temporal relation r^{bi} (in data model M^{bi}) and a bi-temporal instant (c^{tt}, c^{vt}) and returns a non-temporal relation r (in data model M , not containing valid or transaction time) that contains all tuples current at time c^{tt} and valid at time c^{vt} .

Now, one is able to define S -reducibility:

Definition 5. Given non-temporal and temporal data models $M = (DS, QL)$ and $M^{bi} = (DS^{bi}, QL^{bi})$, data model M^{bi} is a syntactically similar snapshot-reducible extension of model M iff

1. data model M^{bi} is snapshot reducible with respect to data model M :

$$\forall q \in QL, \exists q^{bi} \in QL^{bi}, \forall db^{bi} \in DS^{bi}, \forall c^{tt}, c^{vt} : \tau_{(c^{tt}, c^{vt})}^{M^{bi}, M}(q^{bi}(db^{bi})) = q(\tau_{(c^{tt}, c^{vt})}^{M^{bi}, M}(db^{bi}))$$

2. there exist two strings S_1 and S_2 , such that each query $q^{bi} \in QL^{bi}$ that is snapshot reducible with respect to a query $q \in QL$ is syntactically identical to $S_1 q S_2$.

The strings S_1 and S_2 are termed *statement modifiers*.

Example 6. If one evaluates the temporal query

`SEQ VT SELECT COUNT(*) AS "Employees" FROM EmployeeTemp`

on `EmployeeTemp` from the previous example, the result is

Employees	VTIME
2	1999-2000
3	2001-2002
2	2003-2004
1	2005-2006
1	2007-2009

As you can see, evaluating the non-temporal query

`SELECT COUNT(*) AS "Employees" FROM EmployeeTemp`

e.g. on `EmployeeTemp`'s snapshot at time 2002

Name	Salary
Paul	20000
Sarah	30000
Peter	20000

produces the result

Employees
3

and one could therefore evaluate this non-temporal query on every snapshot of `EmployeeTemp` and combine these results to the final temporal result.

2.4 Extended S -reducibility

S -reducibility can only be applied to queries of the underlying non-temporal query language and does not extend to queries that explicitly reference time.

Example 7. Consider the query

```
SEQ VT SELECT COUNT(*), VTIME(EmployeeTemp) AS "Employees" FROM EmployeeTemp
```

In this query, $VTIME(EmployeeTemp)$ explicitly references valid-time data of the argument relation, and it can't be evaluated based on a single (non-temporal) snapshot of the argument relation.

Since such extended queries naturally arise, e.g. when translating a non-temporal query to a temporal one, one needs to extend S -reducibility also to queries that reference the time explicitly, making sure that the presence of explicit time references *does not change the S -reducible evaluation mode of the rest of the statement*.

In order to define extended S -reducibility ([2], page 13), we let $c(r)$ be the constant period $PERIOD\ 'X-Y'$, where X is the smallest and Y the largest time point in relation r . Let $\nu(q)$ be the query where all explicit references to the valid time of a relation r in q are replaced by $c(r)$. Let $[q(db^t)]_{M^t}^\nu$ be the result of evaluating q on db^t in model M^t , with the addition that all explicit references to the valid time of a relation r evaluate to $c(r)$.

Definition 8. Let $M^s = (DS^s, QL^s)$ be a non-temporal data model, and $M^t = (DS^t, QL^t)$ be an S -reducible extension of M^s . Model M^t is an extended S -reducible extension of M^s iff for any S -reducible query $q^t \in QL^t$ it holds that

$$\forall q \in QL^t, \forall db^t \in DS^t : \nu(q) = q^t \implies [q^t(db^t)]_{M^t} = [q(db^t)]_{M^t}^\nu$$

This definition states that if a data model M^t is extended S -reducible to a data model M^s and for a query $q \in QL^t$ there exists a S -reducible query q^t such that $\nu(q) = q^t$, then $[q^t(db^t)]_{M^t} = [q(db^t)]_{M^t}^\nu$, so one can still "evaluate" q on M^s by evaluating the corresponding query q^t on M^s , which can be applied to given snapshots of the underlying temporal database.

2.5 Interval preservation

Another question that arises is whether results of temporal queries should be coalesced based on their timestamps or not. To answer this question, *interval preservation* which requires timestamps are changed as little as possible is introduced in this section.

First, we present some auxiliary definitions. Let Q be a query, $r_1, \dots, r_n$ be relations, and x be a sequence of explicit attribute values of a given result tuple. The *output points* is a formula

$$op(Q, r_1, \dots, r_n, x, A)$$

where $r_1, \dots, r_n$ are the argument relations of Q and A is the unique set of time points associated with the result tuple x , that means

$$op(Q, r_1, \dots, r_n, x, A) \wedge op(Q, r_1, \dots, r_n, x, A') \implies A = A'$$

Example 9. Consider again example 6 on the preceding page. The temporal query

```
q = SEQ VT SELECT COUNT(*) AS "Employees" FROM EmployeeTemp
```

evaluated on `EmployeeTemp` produces

```
op(q, EmployeeTemp, <1>, {2005, 2006, 2007, 2008, 2009})
op(q, EmployeeTemp, <2>, {1999, 2000, 2003, 2004})
op(q, EmployeeTemp, <3>, {2001, 2002})
```

Next, let I be a time interval. The *required argument tuples* are defined as follows:

$$rat(r_1, \dots, r_n, Q, x, I, s_1, \dots, s_n) \iff$$

1. $op(r_1, \dots, r_n, Q, x, A) \wedge I \subseteq A \wedge$
2. $(s_1 \subseteq r_1, \dots, s_n \subseteq r_n) \wedge op(s_1, \dots, s_n, Q, x, A') \wedge I \subseteq A' \subseteq A \wedge$
3. $(\forall s'_1, \dots, s'_n : s'_1 \subseteq s_1, \dots, s'_n \subseteq s_n \wedge \bigcup s'_i \subsetneq \bigcup s_i \wedge op(s'_1, \dots, s'_n, Q, x, A'')) \implies \neg(I \subseteq A'' \subseteq A')$

The required argument tuples are minimal subsets of argument tuples such that the output points still include all the time points of a given interval I . But if we try to reduce the minimal input tuples $s_1, \dots, s_n$ further, the result is either no longer valid for all time points contained in I , or it becomes valid at some additional time points that are not part of the output points of $s_1, \dots, s_n$ (Line 3).

Example 10. From the previous example, one gets

```

rat(EmployeeTemp, q, ⟨1⟩, 2007 – 2009, {⟨Lisa, 25000 | 2007 – 2009⟩})
rat(EmployeeTemp, q, ⟨1⟩, 2007 – 2008, {⟨Lisa, 25000 | 2007 – 2009⟩})
rat(EmployeeTemp, q, ⟨2⟩, 1999 – 2000, {⟨Paul, 20000 | 1999 – 2002⟩, ⟨Sarah, 30000 | 1999 – 2004⟩})
rat(EmployeeTemp, q, ⟨2⟩, 2003 – 2004, {⟨Peter, 25000 | 2003 – 2006⟩, ⟨Sarah, 30000 | 1999 – 2004⟩})

```

As an example,

```

rat(EmployeeTemp, q, ⟨2⟩, 2003 – 2004, {⟨Peter, 20000 | 2001 – 2002⟩, ⟨Peter, 25000 | 2003 – 2006⟩,
                                           ⟨Sarah, 30000 | 1999 – 2004⟩})

```

would have been wrong since $\langle \text{Peter}, 20000 | 2001 - 2002 \rangle$ is not necessary to produce the output tuple during the given interval.

Finally, let $s_1, \dots, s_n$ be required argument tuples. The *maximal impact tuples* are defined as follows:

$$mit(r_1, \dots, r_n, Q, x, I, s_1, \dots, s_n) \iff$$

1. $rat(r_1, \dots, r_n, Q, x, I, s_1, \dots, s_n) \wedge$
2. $\neg \exists I' : (I \subset I' \wedge rat(r_1, \dots, r_n, Q, x, I', s_1, \dots, s_n) \wedge$
3. $\neg \exists I'', s'_1, \dots, s'_n : I'' \subset I \wedge s'_1 \subseteq r_1, \dots, s'_n \subseteq r_n \wedge \bigcup s'_i \subset \bigcup r_i \wedge (s_1 \neq s'_1 \vee \dots \vee s_n \neq s'_n) \wedge$
 $rat(r_1, \dots, r_n, Q, x, I'', s'_1, \dots, s'_n))$

The maximal argument tuples are required input tuples that maximize the interval I . Furthermore, there do not exist alternative required input tuples that permit the interval I to become smaller (line 3).

Example 11. For the ongoing example, we get

```

mit(EmployeeTemp, q, ⟨1⟩, 2007 – 2009, {⟨Lisa, 25000 | 2007 – 2009⟩})
mit(EmployeeTemp, q, ⟨1⟩, 2005 – 2006, {⟨Peter, 25000 | 2003 – 2006⟩})
mit(EmployeeTemp, q, ⟨2⟩, 1999 – 2000, {⟨Paul, 20000 | 1999 – 2002⟩, ⟨Sarah, 30000 | 1999 – 2004⟩})
mit(EmployeeTemp, q, ⟨2⟩, 2003 – 2004, {⟨Peter, 25000 | 2003 – 2006⟩, ⟨Sarah, 30000 | 1999 – 2004⟩})
mit(EmployeeTemp, q, ⟨3⟩, 2001 – 2002,
    {⟨Paul, 20000 | 1999 – 2002⟩, ⟨Sarah, 30000 | 1999 – 2004⟩, ⟨Peter, 20000 | 2001 – 2002⟩})

```

None of the above intervals can be made any smaller by selecting alternative tuples, and they are also as large as possible.

Using these definitions so far, we can formally define interval preservation:

Definition 12. A query Q is interval preserving iff, for all argument tuples, $r_1, \dots, r_n$, the query result, $Q(r_1, \dots, r_n)$, coincides with the maximal impact tuples:

$$\begin{aligned}
&\forall r_1, \dots, r_n, x, I : \langle x | I \rangle \in Q(r_1, \dots, r_n) \\
&\iff \exists s_1, \dots, s_n : mit(r_1, \dots, r_n, Q, x, I, s_1, \dots, s_n)
\end{aligned}$$

Now it is clear that the result of our example query is indeed interval preserving, since every tuple in the result corresponds to a set of maximal impact tuples (see example 6 on page 3). However, if the last two result tuples were coalesced, the result would no longer be interval preserving.

2.6 Non-restrictiveness

The last desideratum is non-restrictiveness ([2], page 18).

Definition 13. A temporal query language is non-restrictive iff it offers statements that manipulate timestamps as regular values, with no built-in temporal semantics enforced, and vice versa.

Example 14. A simple example of a query without temporal semantics is

NSEQ VT SELECT COUNT(*) AS "Employees" FROM EmployeeTemp

Evaluating this query on `EmployeeTemp` produces

Employees
5

since it treats temporal data as regular attributes.

2.7 Summary

In this section, several desiderata have been presented that ensure the proper functioning of a temporally extended query language. The desiderata can be split into two categories:

- *Desiderata facilitating the smooth transition to a temporal DBMS:* upward compatibility, temporal upward compatibility, S-reducibility
- *Desiderata concerning temporal support:* S-reducibility, extended S-reducibility, interval preservation, non-restrictiveness

Based on these desiderata, syntactic and semantic extensions of non-temporal query languages are now being discussed.

3 Extending the syntax of SQL-based languages

Based on the desiderata stated in the previous section, a temporal extension of SQL-92 is presented now ([2], page 20 ff). It serves as an example how to extend a given non-temporal query language, and all examples in the previous section make use of it.

Such an extension must contain all statements of SQL-92 since it has to be upward compatible. Furthermore, all SQL-92 facilities like nested queries or aggregates must be supported. Since SQL-92 statements must work on both temporal and non-temporal relations, SQL-92 statements modify the current and future states of temporal relations, and queries and views consider only snapshot states of argument relations that are current and valid at the times they are evaluated. S-reducibility dictates a modifier that can be prepended to any SQL-92 query to get the corresponding temporal query. For valid and for transaction time, `SEQ VT` and `SEQ TT` are chosen, respectively. Similarly, non-restrictiveness demands a way to control timestamps explicitly. The corresponding modifiers are `NSEQ VT` and `NSEQ TT`, making available the timestamps of temporal relations as regular, explicit attributes.

Based on these considerations, the following syntax is used for extensions at the statement level:

```

<table definitons> ::= "CREATE" "TABLE" <table name>
                        <table element list>
                        ["AS" "VT" ["AND" "TT"]] ["AS" "TT"]

<query expression'> ::= <modifiers> <query expression>
                        | "(" <modifiers> <query expression> ")"
                        <coal>

<assertion check'> ::= <modifiers> <assertion check>

<column constraint'> ::= <modifiers> <column constraint>

<table constraint'> ::= <modifiers> <table constraint>

<SQL data change statement> ::= <modifiers> <insert statement> |
                                <modifiers> <delete statement> |
                                <modifiers> <update statement>

```

Table definitions permit declaration of valid-time, transaction-time or bi-temporal tables. In query expressions, the scope of the semantics implied by the statement modifiers spans the whole query including sub queries, but with the exception of derived table expressions in the from clause which have their own scope. `<coal>` is used to specify a query whose result should be coalesced.

Statement modifiers are specified as follows:

```

<modifiers>      ::= [<modifier> [ "AND" <modifier>]] [<time-range>]
<modifier>       ::= <mode> <dimension> [<time-domain>]
<mode>           ::= "SEQ" | "NSEQ"
<dimension>      ::= "TT" | "VT"
<time-domain>    ::= period_constant
<time-range>     ::= "SET" "VT" period_expression

```

Based on this definitions, the core semantics of temporal statements are as follows [2]:

- No modifier dictates upward compatibility when none of the argument relations support that time. Otherwise, temporal upward compatibility is dictated.
- SEQ dictates evaluation consistent with sequenced semantics. The time dimension will be present in query results.
- NSEQ dictates timestamp processing with no built-in semantics enforced by the DBMS.

It is worthwhile to note that the kind of evaluation can be specified for each time individually.

Time-domain specifications make it possible to restrict the argument database to a certain part that is valid or current during the specified period. Time-range specifications are used to specify the valid time of a query result.

Finally, we need a syntax to specify how a query result will be coalesced:

```

<coal> ::= "(" dimension ")"

```

The semantics of coalescing depends on the type of argument relation. While non-temporal relations cannot be coalesced, valid-time and transaction-time relations can be coalesced, which means that value-equivalent tuples with overlapping or adjacent time periods are merged. Coalescing bi-temporal relations is more subtle. In general, value-equivalent, overlapping tuples do not coalesce into a single tuple, so several result tuples must be generated. These result tuples can be maximized in valid-time or in transaction-time. Furthermore, these two basic operations can be combined, which means that the tuples are first coalesced in one time and then in the other time.

Example 15. To present some more syntax than the examples in the previous section, consider the query

```

(SEQ VT SET VT PERIOD ('1999', END(VTIME(E)))
  SELECT E.Salary FROM EmployeeTemp AS E
  WHERE E.Salary > 15000) (VT)

```

The query selects the tuples from `EmployeeTemp` with a `Salary` greater than 15000, sets valid time to the period from 1999 until `VTIME(E)` (`VTIME(E)` denotes the end of a tuple's valid time) and performs coalescing based on valid time.

4 Defining the formal semantics

Finally, the semantics of a temporal extension are now defined in this section. First, certain new relational algebra operators are defined and temporal statements then translated to well-defined algebraic expressions.

Throughout this section, the following conventions are used: $\langle t \rangle$, $\langle t | VT \rangle$, $\langle t | TT \rangle$ and $\langle t | VT, TT \rangle$ all denote a tuple that stores some information t . The first tuple stores no temporal information, whereas the remaining three tuples store valid-time (VT), transaction-time (TT) and bi-temporal information.

4.1 The temporal relational algebra

4.1.1 Valid-time and transaction-time algebra

In the following table, the definitions of six fundamental relational algebra operators are given ([2], page 29). Since the definitions of the operators are very similar, they are given polymorphic, using t_T and T to denote the type of temporal information and the temporal information itself, respectively. T^+ denotes the starting time of a period T , whereas T^- denotes the end of that period. $\circ$ denotes tuple concatenation, and *intersect* and *overlaps* are operators defined in the next subsection.

$$\sigma_c^{tT}(r) = \{\langle t|T \rangle : \langle t|T \rangle \in r \wedge c(\langle t, T \rangle)\}$$

$$\pi_f^{tT}(r) = \{\langle t_1|T \rangle : \exists t_2 : \langle t_2|T \rangle \in r \wedge t_1 = f(\langle t_2, T \rangle)\}$$

$$r_1 \cup^{tT} r_2 = \{\langle t|T \rangle : \langle t|T \rangle \in r_1 \vee \langle t|T \rangle \in r_2\}$$

$$r_1 \times^{tT} r_2 = \{\langle \langle t_1, T_1 \rangle \circ \langle t_2, T_2 \rangle | T \rangle : \langle t_1|T_1 \rangle \in r_1 \wedge \langle t_2|T_2 \rangle \in r_2 \wedge T = \text{intersect}(T_1, T_2) \wedge T_1 \text{ overlaps } T_2\}$$

$$\begin{aligned} r_1 \setminus^{tT} r_2 = & \{\langle t|T \rangle : \exists T_1 : \langle t|T_1 \rangle \in r_1 \wedge \\ & ((\exists T_2 : \langle t|T_2 \rangle \in r_2 \wedge T_1^- \leq T_2^+ \wedge T^- = \text{succ}(T_2^+)) \vee T^- = T_1^-) \wedge \\ & ((\exists T_3 : \langle t|T_3 \rangle \in r_2 \wedge T_1^+ \geq T_3^- \wedge T^+ = \text{pred}(T_3^-)) \vee T^+ = T_1^+) \\ & \wedge T^- \leq T^+ \wedge \neg(\exists T_4 : \langle t|T_4 \rangle \in r_2 \wedge T_4 \text{ overlaps } T)\} \end{aligned}$$

$$\begin{aligned} \text{coal}^{tT}(r) = & \{\langle t|T \rangle : \exists T_1, T_2 : \langle t|T_1 \rangle \in r \wedge \langle t|T_2 \rangle \in r \wedge \\ & T_1^- < T_2^+ \wedge T^- = T_1^- \wedge T^+ = T_2^+ \wedge \\ & (\forall T_3 : \langle t|T_3 \rangle \in r \wedge T^- < T_3^- < T^+ \implies \\ & (\exists T_4 : \langle t|T_4 \rangle \in r \wedge T_4^- < T_3^- \leq \text{succ}(T_4^+))) \wedge \\ & (\neg \exists T_5 : \langle t|T_5 \rangle \in r \wedge (T_5^- < T^- \leq \text{succ}(T_5^+) \vee \text{pred}(T_5^-) \leq T^+ < T_5^+))\} \end{aligned}$$

Most definitions are straight-forward, except the temporal difference and temporal coalescing, so we will look at them in more detail. The temporal difference are tuples $\langle t|T \rangle$, such that there exists a tuple $\langle t|T_1 \rangle \in r_1$ and two tuples $\langle t|T_2 \rangle, \langle t|T_3 \rangle \in r_2$ reduce the period T since they overlap the period T_1 partially. Alternatively, there do not exist $\langle t|T_2 \rangle, \langle t|T_3 \rangle$, but then, $T^+ = T_1^+$ and $T^- = T_1^-$ must hold. Finally, the forth line states that the period T is properly specified by its starting and ending points and that there is no tuple $\langle t|T_4 \rangle \in r_2$ that completely overlaps the resulting tuple $\langle t|T \rangle$.

The coalescing operator requires the existence of two tuples with the same data and different, but partially overlapping periods (first and second line). Additionally, every tuple $\langle t|T_3 \rangle$ that overlaps the resulting tuple partially must be partially overlapped by another tuple $\langle t|T_4 \rangle$ with $T_4^- < T_3^- \leq \text{succ}(T_4^+)$ to avoid gaps between any tuples coalesced. Finally, there must not exist another tuple $\langle t|T_5 \rangle$ whose period T_5 is not fully contained in T , indicating T being too small.

4.1.2 The bi-temporal algebra

The following bi-temporal operators are natural generalizations of the valid-time and transaction-time operators. Bi-temporal selection, projection, set union and the Cartesian product are straightforward extensions.

$$\sigma_c^{bi}(r) = \{\langle t|VT, TT \rangle : \langle t|VT, TT \rangle \in r \wedge c(\langle t|VT, TT \rangle)\}$$

$$\pi_f^{bi}(r) = \{\langle t_1|VT, TT \rangle : \exists t_2 : \langle t_2|VT, TT \rangle \in r \wedge t_1 = f(\langle t_2|VT, TT \rangle)\}$$

$$r_1 \cup^{bi} r_2 = \{\langle t|VT, TT \rangle : \langle t|VT, TT \rangle \in r_1 \vee \langle t|VT, TT \rangle \in r_2\}$$

$$\begin{aligned} r_1 \times^{bi} r_2 = & \{\langle \langle t_1, VT_1, TT_1 \rangle \circ \langle t_2, VT_2, TT_2 \rangle | VT, TT \rangle : \\ & \langle t_1|VT_1, TT_1 \rangle \in r_1 \wedge \langle t_2|VT_2, TT_2 \rangle \in r_2 \wedge \\ & VT = \text{intersect}(VT_1, VT_2) \wedge TT = \text{intersect}(TT_1, TT_2) \wedge \\ & VT_1 \text{ overlaps } VT_2 \wedge TT_1 \text{ overlaps } TT_2\} \end{aligned}$$

Since the bi-temporal difference and bi-temporal coalescing operators are more complex to define and require additional predicates, their formal definitions are skipped here. They can be found in [1].

4.1.3 Additional algebraic operators

Whereas the operators so far were already known from non-temporal relational algebra, the next three additional operators are new. The first one, τ_{tp} , selects all tuples in the argument relation with a timestamp that overlaps time point tp . The second operator, δ_{per} , selects all those tuples in the argument relation that overlap with period per . Finally, the third operator SN turns a time dimension into an explicit attribute.

The detailed semantics are as follows ([2], page 43):

Function	Semantics, given a non-temporal relation r
$\tau_{tp}^{vt}(r)$	r if $tp = now$, otherwise undefined
$\tau_{tp}^{tt}(r)$	r if $tp = now$, otherwise undefined

Function	Semantics, given a valid-time relation r
$\tau_{tp}^{vt}(r)$	$\{\langle t \rangle : \exists VT : \langle t VT \rangle \in r \wedge VT \text{ overlaps } tp\}$
$\tau_{tp}^{tt}(r)$	r
$\delta_{per}^{vt}(r)$	$\{\langle t VT \rangle : \exists VT' : \langle t VT' \rangle \in r \wedge VT' \text{ overlaps } per \wedge VT = \text{intersect}(VT', per)\}$
$\delta_{per}^{tt}(r)$	r
$SN^{vt}(r)$	$\{\langle t, VT \rangle : \langle t VT \rangle \in r\}$
$SN^{tt}(r)$	r

Function	Semantics, given a transaction-time relation r
$\tau_{tp}^{vt}(r)$	r
$\tau_{tp}^{tt}(r)$	$\{\langle t \rangle : \exists TT : \langle t TT \rangle \in r \wedge TT \text{ overlaps } tp\}$
$\delta_{per}^{vt}(r)$	r
$\delta_{per}^{tt}(r)$	$\{\langle t TT \rangle : \exists TT' : \langle t TT' \rangle \in r \wedge TT' \text{ overlaps } per \wedge TT = \text{intersect}(TT', per)\}$
$SN^{vt}(r)$	r
$SN^{tt}(r)$	$\{\langle t, TT \rangle : \langle t TT \rangle \in r\}$

Function	Semantics, given a bi-temporal relation r
$\tau_{tp}^{vt}(r)$	$\{\langle t TT \rangle : \exists VT : \langle t VT, TT \rangle \in r \wedge VT \text{ overlaps } tp\}$
$\tau_{tp}^{tt}(r)$	$\{\langle t VT \rangle : \exists TT : \langle t VT, TT \rangle \in r \wedge TT \text{ overlaps } tp\}$
$\delta_{per}^{vt}(r)$	$\{\langle t VT, TT \rangle : \exists VT' : \langle t VT', TT \rangle \in r \wedge VT' \text{ overlaps } per \wedge VT = \text{intersect}(VT', per)\}$
$\delta_{per}^{tt}(r)$	$\{\langle t VT, TT \rangle : \exists TT' : \langle t VT, TT' \rangle \in r \wedge TT \text{ overlaps } per \wedge TT = \text{intersect}(TT', per)\}$
$SN^{vt}(r)$	$\{\langle t, VT TT \rangle : \langle t VT, TT \rangle \in r\}$
$SN^{tt}(r)$	$\{\langle t, TT VT \rangle : \langle t VT, TT \rangle \in r\}$

Note that all functions behave like the identity function in cases where the argument relation does not provide the required time information. Also note that the SN function is generally not needed on the implementation level.

4.2 Translating temporal statements to algebra expressions

Based on the work done in the previous subsection, it is now possible to define the semantics of the various constructs in a temporal query language. In this section, $[SQL - 92]_{SQL-92}$ denotes a SQL-92 statement that evaluates to the standard relational algebra expression. $[SQL - 92]_T$, where $T \in \{vt, tt, bi\}$ evaluates to the same algebraic expression as $[SQL - 92]_{SQL-92}$, but with every non-temporal algebraic operator replaced by the corresponding temporal operator.

4.2.1 Constructs for timestamp manipulation

The following table lists some examples of constructs used to work with timestamps. The full table can be found in [2], page 42. In the following examples, tp denotes a time point and iv a time duration. p is a shorthand for $PERIOD\ 'tp_1 - tp_2'$.

Syntax	Semantics
$[PERIOD\ 'tp_1 - tp_2']_{temp}$	$TIMESTAMP\ 'tp_1', TIMESTAMP\ 'tp_2'$
$[FIRST(TIMESTAMP\ 'tp_1', TIMESTAMP\ 'tp_2')]_{temp}$	$\min(tp_1, tp_2)$
$[LAST(TIMESTAMP\ 'tp_1', TIMESTAMP\ 'tp_2')]_{temp}$	$\max(tp_1, tp_2)$
$[BEGIN(p)]_{temp}$	$[FIRST([p]_{temp})]_{temp}$
$[END(p)]_{temp}$	$[LAST([p]_{temp})]_{temp}$
$[p_1\ OVERLAPS\ p_2]_{temp}$	$[END(p_1)]_{temp} \geq [BEGIN(p_2)]_{temp} \wedge [END(p_2)]_{temp} \geq [BEGIN(p_1)]_{temp}$
$[INTERSECT(p_1, p_2)]_{temp}$	$\max([BEGIN(p_1)]_{temp}, [BEGIN(p_2)]_{temp}), \min([END(p_1)]_{temp}, [END(p_2)]_{temp})$

4.2.2 Query expressions

In this subsection, the meaning of temporal query expressions is defined by translating them to well-defined algebraic expressions ([2], page 26):

$$\begin{aligned}
[SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{SQL-92}(\tau_{now}^{tt}(\tau_{now}^{vt}(r_1)), \dots, \\
&\quad \tau_{now}^{tt}(\tau_{now}^{vt}(r_n))) \\
[SEQ VT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{vt}(\tau_{now}^{tt}(r_1), \dots, \tau_{now}^{tt}(r_n)) \\
[NSEQ VT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{SQL-92}(\tau_{now}^{tt}(SN^{vt}(r_1)), \dots, \\
&\quad \tau_{now}^{tt}(SN^{vt}(r_n))) \\
[SEQ TT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{tt}(\tau_{now}^{vt}(r_1), \dots, \tau_{now}^{vt}(r_n)) \\
[NSEQ TT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{SQL-92}(\tau_{now}^{vt}(SN^{tt}(r_1)), \dots, \\
&\quad \tau_{now}^{vt}(SN^{tt}(r_n))) \\
[SEQ VT AND SEQ TT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{bi}(r_1, \dots, r_n) \\
[SEQ VT AND NSEQ TT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{vt}(SN^{tt}(r_1), \dots, SN^{tt}(r_n)) \\
[NSEQ VT AND SEQ TT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{tt}(SN^{vt}(r_1), \dots, SN^{vt}(r_n)) \\
[NSEQ VT AND NSEQ TT SQL - 92]_{temp}(r_1, \dots, r_n) &= [SQL - 92]_{SQL-92}(SN^{tt}(SN^{vt}(r_1)), \dots, \\
&\quad SN^{tt}(SN^{vt}(r_n)))
\end{aligned}$$

This approach can be generalized to other operators and other types of statements.

4.2.3 Domain and range specifications

Next, the meaning of time-domain restrictions must be defined. Such restrictions constrain argument relations to contain only tuples that are valid during a specific period, and only tuples intersecting with the restriction are considered in the query ([2], page 27).

$$\begin{aligned}
&[<mode> VT <time-domain> q^T]_{temp}(r_1, \dots, r_n) \\
&= [<mode> VT q^T]_{temp}(\delta_{<time-domain>}^{vt}(r_1), \dots, \delta_{<time-domain>}^{vt}(r_n)) \\
&\quad [<mode> TT <time-domain> q^T]_{temp}(r_1, \dots, r_n) \\
&= [<mode> TT q^T]_{temp}(\delta_{<time-domain>}^{tt}(r_1), \dots, \delta_{<time-domain>}^{tt}(r_n))
\end{aligned}$$

Furthermore, it is possible to specify time ranges that determine the valid times of the result tuples, using the modifier **SEQ VT range**. If a core statement is a **SEQ VT** statement, the automatically computed valid time is replaced by the value resulting from the time-range specification. For all other core statements, prepending **SEQ VT range** has the effect that the result from evaluating *range* is included into the result as valid time, thus changing the result type since these statements do not produce results containing time information. The detailed specification is as follows ([2], page 28):

$$[SET VT range q^T]_{temp}(r_1, \dots, r_n) = \begin{cases} \left\{ \langle t | VT \rangle : \langle t \rangle \in [q^T]_{temp}(r_1, \dots, r_n) \wedge VT = range(t) \right\} \\ \text{if } [q^T]_{temp}(r_1, \dots, r_n) \text{ evaluates to a nontemporal relation} \\ \\ \left\{ \langle t | VT \rangle : \langle t | VT' \rangle \in [q^T]_{temp}(r_1, \dots, r_n) \wedge VT = range(t) \right\} \\ \text{if } [q^T]_{temp}(r_1, \dots, r_n) \text{ evaluates to a valid-time relation} \\ \\ \left\{ \langle t | VT, TT \rangle : \langle t | TT \rangle \in [q^T]_{temp}(r_1, \dots, r_n) \wedge VT = range(t) \right\} \\ \text{if } [q^T]_{temp}(r_1, \dots, r_n) \text{ evaluates to a transaction-time relation} \\ \\ \left\{ \langle t | VT, TT \rangle : \langle t | VT', TT \rangle \in [q^T]_{temp}(r_1, \dots, r_n) \wedge VT = range(t) \right\} \\ \text{if } [q^T]_{temp}(r_1, \dots, r_n) \text{ evaluates to a bitemporal relation} \end{cases}$$

4.2.4 Coalescing

Let q^T denote a temporal query. If this query returns a valid-time relation, it may be modified to (q^T) (VT), thus returning the coalesced version of its result. The same holds for (q^T) (TT) if q^T returns a transaction-time relation. If q^T returns a bi-temporal relation, it may be coalesced in valid time, transaction time or both time dimensions. The detailed specifications are as follows ([2], page 28):

$$[(q^T) \text{ (VT)}]_{temp}(r_1, \dots, r_n) = \begin{cases} coal^{vt}([q^T]_{temp}(r_1, \dots, r_n)) \\ \text{if } [q^T]_{temp}(r_1, \dots, r_n) \text{ evaluates to a valid-time relation} \end{cases}$$

$$[(q^T) \text{ (TT)}]_{temp}(r_1, \dots, r_n) = \begin{cases} coal^{tt}([q^T]_{temp}(r_1, \dots, r_n)) \\ \text{if } [q^T]_{temp}(r_1, \dots, r_n) \text{ evaluates to a transaction-time relation} \\ coal^{bi}_{vt}([q^T]_{temp}(r_1, \dots, r_n)) \\ \text{if } [q^T]_{temp}(r_1, \dots, r_n) \text{ evaluates to a bitemporal relation} \end{cases}$$

4.2.5 Examples

As a simple example, consider the following non-sequenced query:

NSEQ VT

```
SELECT E.Name, E.Salary FROM EmployeeTemp AS E
WHERE BEGIN(VTIME(E)) = '1999'
```

According to the specifications above, this query is translated to the following relational algebra expression:

$$\pi_{E.Name, E.Salary}(\sigma_{BEGIN(VTIME(E))='1999'}(\tau_{now}^{tt}(SN^{vt}(EmployeeTemp/E))))$$

The similar, but extended query

SEQ VT PERIOD '1999-2002'

```
SELECT E.Name, E.Salary FROM EmployeeTemp AS E
WHERE BEGIN(VTIME(E)) = '1999'
```

is translated to the expression

$$\pi_{E.Name, E.Salary}^{vt}(\sigma_{BEGIN(VTIME(E))='1999'}^{vt}(\tau_{now}^{tt}(\delta_{1999-2002}^{vt}(EmployeeTemp/E))))$$

where the non-temporal algebra operators π and σ are replaced by their valid-time variants π^{vt} and σ^{vt} .

5 Conclusion

This report outlined an approach how to extend a given non-temporal query language with temporal language features that was presented in [2]. First, it put together a set of requirements/desiderata that a temporal extension needs to satisfy. This desiderata ensured a smooth transition to a temporal extended language as well as systematic temporal support by the language and the underlying DBMS.

Based on the desiderata, the paper demonstrated how the syntax and the semantics of a given non-temporal language could be extended. The extension of the syntax was demonstrated at SQL-92 in section 3, other query languages can be extended similarly. The semantics were defined based on a new set of relational algebra operators in section 4 for different types of statements, including range specifications, coalescing and the core semantics of queries. Since these semantics were defined in terms of algebra operators, they can easily be adopted by other query languages.

References

- [1] BOHLEN, M., AND JENSEN, C. Seamless integration of time into SQL. *ACM Transactions on Database Systems* (1996).
- [2] BÖHLEN, M. H., JENSEN, C. S., AND SNODGRASS, R. T. Temporal statement modifiers. *ACM Trans. Database Syst.* 25, 4 (2000), 407–456.