ANSI X3H2-95-487
ISO/IEC JTC1/SC21/WG3 DBL LHR-?

I S O
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION

December 6, 1995

Subject: SQL3 Part 7: Temporal

Status: ANSI Expert's Contribution

Title: Response to LHR-043, "Fixing possible problems in SQL/T"

Version: 1

Author: Richard T. Snodgrass

Abstract: This document evaluates the changes proposed in LHR-043.

References:

1	Valid-Time Support in SQL3	
2	Collection type Range	YOW-025
3	Comments on SQL/Temporal	YOW-155
4	Proposal for a new SQL Part — Temporal	RIO-075
5	SQL/Temporal	LHR-009
6	Possible problems in SQL/T	LHR-042
7	Fixing some possible problems in SQL/T	LHR-043
8	Response to LHR-042	ANSI X3H2-95-486
9	The TSQL2 Temporal Query Language	Kluwer
10	Maintaining Knowledge about Temporal Elements	CACM 11/83

1 Introduction

The abstract for LHR-043 states that that document "proposes a number of changes to SQL/T that are intended to remove some of the criticisms of YOW-125 that were listed in YOW-155 and represented as possible problems in LHR-042." As X3H2-95-486 shows, addressing the problems listed in LHR-042 does not require extensive changes to LHR-009. LHR-043 in fact proposes a series of more substantial additions and changes that are unrelated to LHR-042. In particular, it shares with YOW-025 the generalization of period to allow "the possibility of having periods (or ranges, or whatever name is considered appropriate for such things) of other data types in the future."

2 Ranges and Time

To differentiate 'such things' from periods as defined in LHR-009, let's call them by their original term: *range*. A range is a convex set of values, say integers. A specific integer denotes a quantity of something. Consider liters of milk. The integer 4 might denote 4 liters of milk. The refrigerator might contain a range of 2 to 4 liters of milk.

A period is an anchored duration of time. A particular period, say 3:30pm Friday December 1, 1995 through 5:27 Monday, December 4, 1995, denotes specific days, hours, and minutes. *Unanchored* durations are SQL intervals, an example being 3 days, 1 hour, and 57 minutes, or equivalently, 259317 minutes. SQL intervals can be added to SQL datetimes to offset them to a new (anchored) datetime.

The analogy of ranges to time is quite interesting. A data type like integer or float is most similar to a time interval. One can have 4 liters of milk or 4 days of time. A range of a data type, such as 2 to 4 liters of milk, corresponds to an *indeterminate interval* [1,9], such as 2 to 4 days. We're not sure how much milk our refrigerator contains, just as we're not sure precisely how long the car's radiator fluid has been low.

Let's now consider integers representing temperatures. The temperature 7°C could represent either an absolute temperature (i.e., cold, but not freezing) or could represent an increment of thermal entropy, which could be added to an absolute temperature, say 15°C to get another absolute temperature, 22°C. This analysis concludes that an integer representing a temperature can either be anchored, and thus analogous to a datetime, or be unanchored, and thus analogous to an interval. Considering ranges of temperatures, the range 7°C – 11°C could be analogous to an *indeterminate datetime* [1,9] (if the context is "how cold is it outside?"), an indeterminate interval (if the context is "the temperature increased by 7°C to 11°C over the course of the afternoon), or a period (if the context is "the temperature varied from a low of 7°C to a high of 11°C today").

Indeterminate datetimes and intervals are highly useful in temporal databases. They are fully supported in TSQL2 [1,9]. They may be associated with probability distribution functions, which would also be helpful with ranges (e.g., there is probably only 2 or 3 liters of milk in the refrigerator).

In any case, ranges, being independent of data type, properly belong in SQL/Foundation, not in SQL/Temporal. We encourage the authors to retarget their proposal to that base document. Also, it would be helpful to explore the relationship between ranges and indeterminate datetimes and intervals more fully.

The remainder of the present document will consider what problems and solutions LHR-043 brings to the specific issue of the period data type as defined in SQL/Temporal, as separate from the consideration of the range generalization. It turns out that some of the functionality proposed in LHR-043 is already present in TSQL2, in more generality. TSQL2 represents a consistent and integrated approach to these issues. Some of the suggestions in LHR-043 has been incorporated (and sometimes generalized) in the changes proposed in X3H2-95-486.

The comments will apply to the specific sections of LHR-043, using an identical numbering.

3 PP 1, Definitions

Discussing the "beginning of the granule" and a "(perfectly accurate) clock" (which cannot exist) implies a continuous ontology of time. [1,9] contain careful discussions of why TSQL2, and hence SQL/Temporal, rejects choosing that particular time ontology.

The time ontology used in a language definition interacts closely with granularity and indeterminacy issues, neither of which are covered in any detail in LHR-043. TSQL2, on the other hand, incorporates comprehensive support for both, consistent with the time ontology in SQL/Temporal.

As LHR-043 provides no reason why the time ontology in SQL/Temporal is deficient, there is no compelling reason to change it.

4 PP2, 3, 4, 7, 10, and 11

Defining periods in terms of DATE, TIME, and TIMESTAMP add the ability to specify periods with a precision of DAY. TSQL2 supports all granularities for the precision of a period. As one example, in TSQL2 one can define MONTH periods.

4.1 Clause 4.2

Most of this concerns defining periods as a data type constructor, to later support general ranges.

4.2 Clause 6.1

This concerns ranges.

4.3 Clause 6.3

NEXT and PREVIOUS are general operators, and so are more appropriate for SQL/Foundation. The specific need for NEXT and PREVIOUS on datetime values and LAST on periods is generally obviated in TSQL2 via open-closed periods, as discussed in more detail below.

4.4 Clause 6.4

A major advance in temporal databases was the recognition that periods were not closed under set operations. This insight led to the definition of *temporal elements*, or finite unions of periods, which *were* closed under set operations. Many temporal data models and query languages utilize temporal elements.

P_UNION does not in fact yield a union of two periods, if the periods are disjoint. P_EXCEPT does not in fact yield the difference of two periods, if one period contains another. These operators imply that periods are sets, when in fact they are contiguous granules, not general sets.

4.5 Clause 7.1

UPTO and THROUGH allow periods to either contain or not contain their ending delimiting timestamp. In interval calculus terminology, these reserved words support closed-closed and close-open intervals. TSQL2 allows those two variants, as well as the other two variants, open-closed and open-open (which do not contain their beginning delimiting timestamp), for literals. If these variants are also desired via the PERIOD value constructor, it is possible to do so by exploiting the accepted formalism of "[...]" (for closed-closed), "[...)" (for closed-open), "(...]" (for open-closed), and "(...)" (for open-open).

4.6 Clause 7.2

The suggestions have been adopted, in X3H2-95-486.

4.7 Clause 10.2

This concerns ranges.

4.8 Clause 11.1

These changes are connected with those concerning clause 6.4, above.

5 PP5: Clause 5.2 <literal>

These changes also concern the open-closed distinction, but for literals, which is already handled in a more general fashion in TSQL2.

6 PP6 Clause 6.2 <set function specification>

Clause 6.2 cannot be deleted as proposed because SUM and AVG are not permitted on periods. However, specifying the ordering of periods does simplify things, and so GR1–3 can indeed be removed, as specified in ANSI X3H2-95-486.

7 PP8 and PP9: 6.5 <cast specification>

These "corrections" are actually extensions to support ranges.

8 PP15 7.6 <meets predicate>, and others

This proposal reduces the expressive power in SQL/Temporal. As a specific example, the proposal eliminates `PERIOD '...' MEETS DATE '...'`.

X SUCCEEDS Y is equivalent to Y PRECEDES X. We see no compelling need for another reserved word that adds so little. Allen [10] defines thirteen interval operators, all of which can be expressed using PRECEDES. One could expand the list of reserved words quite a bit to add these operators, with no increase in expressive power.

9 PP12, PP13, and PP14

This proposal reduces the expressive power in SQL/Temporal. As a specific example, the proposal eliminates `PERIOD '...' OVERLAPS DATE '...'`.

10 Expanding a period into a set

Even if a function in useful in defining other operations, that does not imply that the function itself should be included in the language.

11 Summary

Concerning LHR-043, our reactions to these proposals can be summarized as follows.

- Ranges should be retargeted for SQL/Foundation, and the analogy with indeterminate time intervals and datetimes exploited.
- The existing time ontology in SQL/Temporal should be retained.
- If further flexibility in specifying different granularities for the precision of a period is desired, TSQL2 provides much more generality than does LHR-043.
- NEXT, PREVIOUS, LAST, UPTO and THROUGH can all be supported in a manner more consistent with the accepted formalism of interval calculus by adopting TSQL2's approach, which also is more general.
- Set operators on periods encourages the invalid notion that periods are general sets.
- Some of the proposed changes reduce the expressive power of SQL/Temporal, without justification.

- Some of the new operators, such as SUCCEEDS and EXPAND, are insufficiently motivated.

There was one extension proposed that goes beyond the constructs in TSQL2: it might be useful for the PERIOD constructor to indicate which type of period is desired, in a manner similar to period literals.