ANSI X3H2-95-486
ISO/IEC JTC1/SC21/WG3 DBL LHR-092

I S O
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION

December 29, 1995

Subject: SQL3 Part 7: Temporal

Status: Change Proposal

Title: Response to LHR-042, "Possible problems in SQL/T"

Version: 2

Source: ANSI Expert's Contribution

Author: Richard T. Snodgrass

Abstract: This change proposal addresses the concerns listed in LHR-042. It also addresses the possible problems with SQL/Temporal listed in LHR-009, numbered **TMPRL-001** through **TMPRL-012**, as well as the response in LHR-099.

References:

1	Valid-Time Support in SQL3	
2	Comments on SQL/Temporal	YOW-155
3	Proposal for a New SQL Part — Temporal	RIO-075
4	SQL/Bindings	LHR-007
5	SQL/Temporal	LHR-009
6	Possible Problems in SQL/T	LHR-042
7	Fixing some possible problems in SQL/T	LHR-043
8	Clarify assignment rules for datetimes	LHR-046
9	Response to LHR-043	LHR-093
10	The TSQL2 Temporal Query Language	Kluwer
11	SQL/T as amended by LHR-033 and LHR-043	LHR-058
12	Response to LHR-92/93 re SQL/Temporal	LHR-099

LHR-042 is a recasting of the comments that originally appeared in YOW-155. In LHR-042, the problems are grouped into sixteen categories.

In this document, we address these problems, following the same numbered categories. In many cases, the problems are eliminated with changes to the base document, indicated with

> In Section ..., replace "..." with "...".

Some of these changes were suggested by LHR-043, in a different form.

This document should be read with LHR-042 in hand, as we do not repeat many of the statements made in that document.

We then address the possible problems with SQL/Temporal listed in LHR-009, numbered **TMPRL-001** through **TMPRL-012**.

This document appeared as a late paper, LHR-092. Subsequent, Hugh Darwen and J.M. Sykes produced a short response, which was emailed to `isodbl` and submitted as LHR-099.

Version 2 of the present document also responds to LHR-099 and to other minor concerns that have raised, thereby covering all known objections to the original base document, LHR-009. Note that changes were made in several places of version 1 to arrive at version 2 of this document. This was done to collect in one place the accumulated changes to the base document. The specific differences between version 1 and version 2 of this document are as follows.

- The References have been extended with References 11 and 12.
- Ordering of periods: '<' is now defined separately from PRECEEDS.
- <cast specification>: This was modified to correctly handle different precisions.
- Duration of a period: As requested, this was moved out of CAST and made a separate function. The specifics were added to the appendix as <interval value function>.
- <overlaps predicate>: result: We now agree, and have changed the definition to be symmetric.
- The predicates in the appendix have been changed in a minor way to ensure that the result is correct when period values occur in such predicates.
- We have added a new section discussing the concerns raised in LHR-099.
- We have added a new section dealing with other minor concerns that have been raised.

As a result, this document incorporates all the consensual items that have emerged during the interaction between the ANSI and UK DBL committees over the last month.

1 Definitions

- *"P is not specified"* In Section 3.1, replace "**period**: a set of contiguous granules of a specified precision." with "**period**: a set of contiguous granules of a specified precision P."
- *"it is not made clear whether it is a quantum period, or an instant."* This question has far-ranging philosophical implications. Philosophers have argued for ages whether time is discrete (in which time is viewed as being isomorphic to the integers, which each integer coresponding to a "point" in time), dense (in which time is viewed as being isomorphic to the rationals), or continuous (in which time is viewed as being isomorphic to the real numbers). Science and metaphysics have uet to determine which model best fits reality; good arguments can be made for each of the three models. In terms of the TSQL2 language, we view this choice as largely unimportant; time can be either continuous, dense, or discrete. However, since we are modeling time on a discrete computing device, our "view" of time and the uses to which we put that view must necessarily be discrete. Hence, we speak in terms of "granules" instead of instants. This topic is discussed in detail in [1,10].

2 Extensibility

- *"The syntax of <data type>, when used to specify a period ... seems unnecessarily restrictive that there is no provision for periods corresponding to the other SQL datatime data types, DATE and TIME."* TSQL2 contains provision to specify the range and precision of a period, and encompasses DATEs, i.e., periods with a precision of DAY, and TIME, i.e., periods with a range of 24 HOURs and a precision of SECOND. [1,10] give the details. The extended PERIOD declaration syntax will be included in a separate change proposal, where it will be integrated in a consistent fashion with calendars other than the default Gregorian calendar.

  The use of a data type generator is limited to defining period analogues of TIMESTAMP, TIME, and DATE. It does not allow one to specify other ranges, such as a century (instead of the 99 centuries in DATE and TIMESTAMP) nor other ranges, such as MONTH.

3 Assignability and comparability

- *"This interpretation, if correct, it is inconsistent with the approach usually followed in SQL."* As noted in LHR-046 [8], the same statement can be made concerning SQL's datetime types. The intent was that the period type would behave identically to the datetime types in this aspect. When the semantics of datetime types has been resolved, the same changes should be applied to the period type. In the meantime, the SQL/Temporal base document is consistent with the SQL/Foundation base document.

4 Ordering of periods

- *"There is no specification of a less-than operation for values of data type period."* This is an excellent point. While Section 7.4 "<precedes predicate>" of the SQL/Temporal base document provides one such operation, that operation is not appropriate, as it does not provide a total order on a set of periods. We extend Section 7.2 "<comparison predicate>" of the base document of SQL/Temporal to allow period to be compared fully with other periods. First, SR1 of Section 7.2 "<comparison predicate>" is removed. Second, the following item is added to GR1a of Section 7.2 "<comparison predicate>".

  "v) The comparison of two periods is determined by first comparing their beginning delimiting timestamps, and then if necessary, their ending delimiting timestamps. The result of $X < Y$ is defined as the result of the following expression:

  $XS < YS$ OR (($XS = YS$ AND $XE < YE$))"

5 5.2 literal

- *"GR1, last few words are "...as the period"; what period?"* The delimiting timestamps must have the same precision, as specified by SR3 of that section. In Section 5.2, replace GR1 "The beginning and ending delimiting timestamps of a period shall have the same precision as the period." with "The period literal shall have a precision equal to that of the beginning delimiting timestamp.".

- *"GR4...should be moved to 6.1 <data type>...It should also be amended to prohibit null delimiting timestamps."* We agree. GR4 of Section 5.2 "<literal>" should be moved to Section 6.1 "<data type>, and the following sentence added to that general rule: "If the beginning delimiting timestamp or the ending delimiting timestamp is null, then an exception condition is raised: *data exception — invalid period format*."

6 6.2 <set function specification>

- *"Period is the only data type for which MIN or MAX may return a value that is not in the grouped* *table. Specification of ordering would remove this anomaly."* Ordering is specified in GR2 using precedes. In Section 6.2, remove SR2 and GR1–3.

7 6.4 <period value function>

- *"GR2f is redundant."* We agree. GR2f should be discarded.

8 6.5 <cast specification>

- *"The ability to cast from period to DATE or TIMESTAMP seems of small utility, especially in comparison with some of the features left out. Moreover, it does not conform to the generally accepted meaning of 'cast', as 'convert to the equivalent value of some other data type'."* While it may be of small utility, it was included for completeness. The definition seems to be a natural one, and attempts to capture what the "equivalent" DATE or TIMESTAMP is for a given period.

9 6.5 <cast specification>

- *"[the term 'duration'] is not defined"* Each period contains at least one granule, so the duration of the smallest period literal is one granule in the precision of that literal. However, we agree that the use of the term 'duration' is confusing.

  In Section 6.5 "<cast specification>, replace GR1c) with the following.

  "If SD is the datetime date type TIMESTAMP with precision SP, and the precision of TD is TP then

  Case:

  i) If $SP \geq TP$, the conversion results in the value PERIOD(CAST(SV AS TIMESTAMP(TP)), CAST(SV AS TIMESTAMP(TP))).

  ii) If $SP < TP$, the conversion results in the value PERIOD(CAST(SV AS TIMESTAMP(TP)), CAST(SV + INTERVAL '1' SP AS TIMESTAMP(TP)) - INTERVAL '1' TP)."

10 Duration of a period (<interval value expression>?)

- *"There is no explicit way to obtain the value of the interval that represents the duration of a period."* This was an omission; that capability was present in RIO-75, in CAST. However, LHR-099 argues persuasively that using CAST is not the best approach. At the same time, we find PERIOD_LENGTH of LHR-043 to also be problematic, as the other reserved words containing '_' (specifically BIT_LENGTH, CHAR_LENGTH, CHARACTER_LENGTH, and OCTET_LENGTH) all refer to the number of *characters* in the string value. Just as PERIOD is both a data type and a constructor, we prefer to use INTERVAL in exactly the same way. (Another reasonable alternative is DURATION, but that adds a reserved word.) See the Appendix for a new subclass, "<interval value function>".

11 7.2 <comparison predicate>

- *"In General Rule 1a iv, ' "X - Y" ' should presumably read ' "X = Y" '."* We agree. The October, 1995 version (LHR-009) [5] has already incorporated that correction.

12 7.3 <overlaps predicate>: incompatability

- *"See the Editor's note."* This is indeed incompatible. The intent was to extend the allowable arguments to OVERLAPS to include as an argument a period, a single datetime, a periods denoted with a pair of datetimes or period denoted with a datetime-interval pair, with the one exception of both arguments being single datetimes not allowed.

  See the Appendix for a replacement of the entire Section 7.3 "<overlaps predicate>".

13 7.3 <overlaps predicate>: data type of <row value expression>

See the Appendix for a replacement of subclauses 7.3, 7.4, 7.5 and 7.6.

14 7.3 <overlaps predicate>: result

- *"We would expect the result to be true also in the case that B1 >= E2 AND B2 >=E2."* We agree. The appendix now treats the two arguments symmetrically.

15 7.6 <meets predicate>

We agree. See the appendix.

16 Other Parts of the SQL Standard

- *"There are no provisions for dealing with items of data type period in SQL/Bindings."* This requires several additions to SQL/Bindings [4].

  - Add entry PERIOD, with Code 11, to Table 2 "Codes used for SQL data types in Dynamic SQL".
  - Rename Table 3 to "Codes associated with datetime and period data types in Dynamic SQL". Add entries PERIOD, with Code 6, and PERIOD WITH TIME ZONE, with Code 7.
  - In Section 12.1 "Description of SQL item descriptor areas", add to SR3: "n) TYPE indicates a <period type>, DATETIME_INTERVAL_CODE is a code specified in Table 3, "Codes associated with datetime and period data types in Dynamic SQL", and PRECISION is a valid value for the <period precision> of the indicated period datatype.
  - In Section 12.5 "<set desriptor statement>", add to GR5: "n) If V indicates PERIOD, or PERIOD WITH TIME ZONE, then PRECISION is set to 6 and all other item descriptor area fields are set to implementation-dependent values."
  - In Section 12.9 "<using clause>", add to GR3.f.v, GR4.f.v and GR5.f.v: "12) If TYPE indicates a <period type>, then LENGTH is set to the length in positions of the period type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 3, "Codes associated with datetime and period data types in Dynamic SQL", to indicate the specific datetime data type, and PRECISION is set to the <period precision>, if applicable." Add to GR6.f.v: "9) If TYPE indicates a <period type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 3, "Codes associated with datetime and period data types in Dynamic SQL", to indicate the specific datetime data type, and PRECISION is set to the <period precision>, if applicable."

17 Problems Listed in LHR-009

This section considered the possible problems listed in the SQL/Temporal base document, and numbered **TMPRL-001** through **TMPRL-012**. As these problems were primarily drawn from YOW-155, there is substantial duplication with the problems discussed above. Nevertheless, we include all of them here for completeness.

17.1 TMPRL-001

See Section 16 of this document.

17.2 TMPRL-002

See Section 3 of this document.

17.3 TMPRL-003

See Section 4 of this document.

17.4 TMPRL-004

See Section 5 of this document.

17.5 TMPRL-005

While the recommended change is not consistent with the syntax rules for other data types, we agree with the desire for generality. Replace SR3 of Section 6.1 "<data type>" with "3) The length of a period is the number of characters sufficient to hold the longest result of CAST(whatever TO CHARACTER) for a value of the data type."

17.6 TMPRL-006

See Section 6 of this document.

17.7 TMPRL-007

See Section 7 of this document.

17.8 TMPRL-008

See Section 8 of this document.

17.9 TMPRL-009

See Sections 9 and 10 of this document.

17.10 TMPRL-0010

See Section 11 of this document.

17.11 TMPRL-0011

See Sections 12 and 13 of this document.

17.12 TMPRL-00124

See Section 15 of this document.

18 Problems Listed in LHR-099

LHR-099 is a response to the version 1 of the present document. Here we respond to each section of LHR-099.

18.1 A possible misunderstanding

There in fact was no misunderstanding, and we all agree that (a) open-closed as well as closed-closed *representations* of literals are desirable, and (b) this aspect is orthogonal to the implementation of period values within the DBMS.

18.2 Extensibility

The discussion in LHR-099 does not clarify whether the semantics of ranges is (1) a single indeterminate value of the underlying type, (2) an indeterminate interval of values of the underlying type, or (3) an anchored duration of values of the underlying type. This is critical. Using ranges to represent all three is semantically awkward at best, because different operations apply to each semantics. For example, an indeterminate interval of values can be multiplied by a scalar, but it doesn't make sense to multiply the other two types by a scalar. This distinction is not between time and other ranges, rather it is between different kinds of time, or temperature, or whatever. For time, the respective types are (1) indeterminate datetime, (2) an indeterminate interval, and (3) a period.

18.3 Casting PERIOD (0) to PERIOD (n)

This point is well taken: the semantics of this operation were not sufficiently specific. This concern has been addressed in Section 8 "6.5 <cast specification>".

18.4 Duration of a period

Using the new syntax for converting a period into an interval, the example from LHR-099 can be expressed as `INTERVAL(P0) = CAST(INTERVAL(P0) AS INTERVAL SECOND(3))`. The left hand side returns INTERVAL '2' SECOND (since the period consists of two seconds, 00:00:00 and 00:00:01). The right hand side evaluates to INTERVAL '2.000' SECOND(3). So it appears that the new definition of INTERVAL() in the appendix satisfies this concern.

18.5 <cast specification>

We agree that casting from a period to a timestamp is not orthogonal, and are content to remove that case. To do so, remove GR2 from Section 6.5 "<cast specification> and change the appropriate two cells in the table of that section (SD = PERIOD, TD = TIME and TD = TIMESTAMP) from "Y" to "N".

We agree that the use of CAST of PERIOD to INTERVAL is somewhat ad hoc, and are comfortable with a new constructor. See Section 10 "Duration of a period" and the appendix of this document.

18.6 Comparison of periods

The '<' operator now has the desired semantics.

18.7 LHR-009 6.4 <period value function>

These are valid points. Replace the last sentence of GR1 in Section 6.4 "<period value function>" with the following.

"The value of the <period value function> is the null value if either BL or EL is null. Otherwise, the value of the <period value function is the value of:

`PERIOD '[`BL `-` EL`]'`."

Similarly, replace the last sentence of GR2e in Section 6.4 "<period value function>" with the following.

"The value of the <period value function> is the null value if either BL or EL is null. Otherwise, the value of the <period value function is the value of:

`PERIOD '[`BL` - `EL`]'`."

Also, replace "FT with "BT" in the second sentence of GR1.

18.8 Notation, and syntax chosen

The limitation of UPTO and THROUGH is that, while they differentiate between closed-closed and closed-open period literals, they do not allow the symmetric open-closed and open-open period literals. Additionally, they are inconsistent with standard mathematical interval theory formalism and require two new reserved words, neither of which are mentioned in LHR-058. The following proposal does not exhibit any of these disadvantages. In summary, the following four literals all denote the same underlying period:

- `PERIOD '[1995-01-01 00:00:00 - 1995-01-01 23:59:59]'`
- `PERIOD '[1995-01-01 00:00:00 - 1995-01-02 00:00:00)'`
- `PERIOD '(1994-12-31 23:59:59 - 1995-01-01 23:59:59]'`
- `PERIOD '(1994-12-31 23:59:59 - 1995-01-02 00:00:00)'`

Replace the <period string> production in Section 5.2 "<literal>" with the following.

<period string> ::=
<quote> <left period delimiter>
<beginning timestamp> <space> <minus sign> <ending timestamp>
[<time zone interval>]
<right period delimiter> <quote>

Add the following productions.

<left period delimiter> ::=
<left bracket> | <left paren>

<right period delimiter> ::=
<right bracket> | <right paren>

Replace General Rule 2 with the following.

"2) Let A and B be valid <date value><time value> pairs, representing timestamps C and D. Let P be the precision of A. Let L and R be valid <left period delimiter> and <right period delimiter>, respectively. If L is <left bracket>, let $BDT = C$; otherwise, let $BDT = C +$ INTERVAL '1' P. If R is <right bracket>, let $EDT = D$; otherwise, let $EDT = D$ - INTERVAL '1' P. If the <period string> of a <period literal> is identical to

`'`LA` - `BR`'`

or

`'`LA` - `BR <timezone interval>`'`

then the value of the <period literal> is the value of the following:

PERIOD(BDT, EDT)"

The PERIOD constructor could also be thus extended. However, it is first useful to determine if the above syntax is acceptable.

18.9 Verbosity

We agree that the replacement sections in Appendix A are rather verbose, and welcome suggestions for how to factor out the common prose.

18.10 Alleged superfluity in LHR-043

The major problem with P_UNION and P_EXCEPT is that when viewed as set operations, they can return a set of two periods when applied to single periods. Such is not the case with INTERSECT, which is why this operation is permitted in LHR-009.

18.11 Allowing other data types as arguments to predicates

We welcome identification of other data types that could appear as arguments to predicates.

18.12 Points of aggrement

The above sections have identified other points of agreement. We do seem to be converging!

19 Other Concerns

LHR-009 has been shown to others, who have also raised some minor concerns. We address these here.

- In the second sentence of GR4 in Section 5.2 "<literal>", remove "or operation".
- In GR2 of Section 6.5 "<cast specification>, replace "If *SV* is the PERIOD data type" with "If *SD* is the PERIOD data type".
- In GR2f of Section 6.4 "<period value function>", replace "*data exception — invalid period format*" with "*data exception — invalid period value for intersect*". Add to Table 2–SQLSTATE class and subclass values in Section 11.1 "SQLSTATE" the Subcondition under data exception "invalid period value for intersect" with a Subclass of "033".
- In SR1 of Section 6.7 "<period value expression>", replace "period" with "PERIOD".

20 Summary

This change proposal has made some specific change recommendations, thereby addressing all known concerns related to LHR-009. We thank Mike Sykes, Hugh Darwen, and John Bair for their careful reading of LHR-009 and successive change proposals.

A Replaced Sections

The following sections should replace Sections 7.3, 7.4, 7.5 and 7.6, respectively. They generalize these operators to apply to all reasonable combinations of period, datetime, and row value expressions.

A.1 <overlaps predicate>

Function
Specify a test for an overlap between two periods, or between a period and a datetime.
Format

<overlaps predicate> ::=
    <period or datetime value expression 1> `OVERLAPS` <period or datetime value expression 2>

<period or datetime value expression 1> ::= <period or datetime value expression>
<period or datetime value expression 2> ::= <period or datetime value expression>

<period or datetime value expression> ::= <row value expression>
    | <period value expression>
    | <datetime value expression>

Syntax Rules

1. Let <period or datetime value expression 1> be PD_1 and let <period or datetime value expression 2> be PD_2.
2. If PD_1 is <row value expression>, then:
   a) Let R_1 be the <row value expression> in PD_1.
   b) The degree of R_1 shall be 2.
   c) Let T_1 be the type of the first column of R_1, and T_2 be the type of the second column of R_1. T_1 shall be a datetime data type.
   d) If T_2 is a datetime data type, then it shall be comparable with T_1.
   e) If T_2 is INTERVAL, then the precision of T_2 shall be such that the interval can be added to T_1.
   f) The precision of PD_1 is the precision of T_1.
3. If PD_1 is <period value expression> or <datetime value expression>, then the precision of PD_1 is that of the <period value expression> or the <datetime value expression>.
4. If PD_2 is <row value expression>, then:
   a) Let R_2 be the <row value expression> in PD_2.
   b) The degree of R_2 shall be 2.
   c) Let T_3 be the type of the first column of R_2, and T_4 be the type of the second column of R_2. T_3 shall be a datetime data type.
   d) If T_4 is a datetime data type, then it shall be comparable with T_3.
   e) If T_4 is INTERVAL, then the precision of T_4 shall be such that the interval can be added to T_3.
   f) The precision of PD_2 is the precision of T_3.
5. If PD_2 is <period value expression> or <datetime value expression>, then the precision of PD_2 is that of the <period value expression> or the <datetime value expression>.
6. The precision of PD_1 and PD_2 shall be identical. Let this precision be P.

7. PD_1 and PD_2 shall not both be datetime data types.

Access Rules
No additional Access Rules.
General Rules

1. Case:
   (a) If the data type of PD_1 is DATE, TIME, or TIMESTAMP, then let B_1 and E_1 both be the value of PD_1.
   (b) If the data type of PD_1 is PERIOD, then let B_1 be the value of BEGIN(PD_1) and E_1 be the value of END(PD_1) + INTERVAL '1' P.
   (c) Otherwise, let R_1 be the value of <row value expression 1> and let B_1 be the value of the first column of R_1.
   Case:
      i) If the data type of the second column of R_1 is a datetime data type, then let E_1 be the value of the second column of R_1.
      ii) If the data type of the second column of R_1 is INTERVAL, then let I_1 be the value of the second column of R_1. Let $E_1 = B_1 + I_1$.
2. If B_1 is the null value or if $E_1 < B_1$, then let $S_1 = E_1$ and let $T_1 = B_1$. Otherwise, let $S_1 = B_1$ and let $T_1 = E_1$.
3. Case:
   (a) If the data type of PD_2 is DATE, TIME, or TIMESTAMP, then let B_2 and E_2 both be the value of PD_2.
   (b) If the data type of PD_2 is PERIOD, then let B_2 be the value of BEGIN(PD_2) and E_2 be the value of END(PD_2) + INTERVAL '1' P.
   (c) Otherwise, let R_2 be the value of <row value expression 2> and let B_2 be the value of the first column of R_2.
   Case:
      i) If the data type of the second column of R_2 is a datetime data type, then let E_2 be the value of the second column of R_2.
      ii) If the data type of the second column of R_2 is INTERVAL, then let I_2 be the value of the second column of R_2. Let $E_2 = B_2 + I_2$.
4. If B_2 is the null value or if $E_2 < B_2$, then let $S_2 = E_2$ and let $T_2 = B_2$. Otherwise, let $S_2 = B_2$ and let $T_2 = E_2$.
5. The value of the <overlaps predicate> is the result of the following expression:
   ($S_1 > S_2$ AND NOT ($S_1 \geq T_2$ AND $T_1 \geq T_2$))
   OR
   ($S_2 > S_1$ AND NOT ($S_2 \geq T_1$ AND $T_2 \geq T_1$))
   OR
   ($S_1 = S_2$)

A.2 <precedes predicate>

Function
Specify a test for one period or datetime preceding a second period or datetime.
Format

<precedes predicate> ::=
<period or datetime value expression 1> `PRECEDES` <period or datetime value expression 2>

Syntax Rules

1. Let <period or datetime value expression 1> be PD_1 and let <period or datetime value expression 2> be PD_2.
2. If PD_1 is <row value expression>, then:
   a) Let R_1 be the <row value expression> in PD_1.
   b) The degree of R_1 shall be 2.
   c) Let T_1 be the type of the first column of R_1, and T_2 be the type of the second column of R_1. T_1 shall be a datetime data type.
   d) If T_2 is a datetime data type, then it shall be comparable with T_1.
   e) If T_2 is INTERVAL, then the precision of T_2 shall be such that the interval can be added to T_1.
   f) The precision of PD_1 is the precision of T_1.
3. If PD_1 is <period value expression> or <datetime value expression>, then the precision of PD_1 is that of the <period value expression> or the <datetime value expression>.
4. If PD_2 is <row value expression>, then:
   a) Let R_2 be the <row value expression> in PD_2.
   b) The degree of R_2 shall be 2.
   c) Let T_3 be the type of the first column of R_2, and T_4 be the type of the second column of R_2. T_3 shall be a datetime data type.
   d) If T_4 is a datetime data type, then it shall be comparable with T_3.
   e) If T_4 is INTERVAL, then the precision of T_4 shall be such that the interval can be added to T_3.
   f) The precision of PD_2 is the precision of T_3.
5. If PD_2 is <period value expression> or <datetime value expression>, then the precision of PD_2 is that of the <period value expression> or the <datetime value expression>.
6. The precision of PD_1 and PD_2 shall be identical. Let this precision be P.

Access Rules
No additional Access Rules.
General Rules

1. Case:
   (a) If the data type of PD_1 is DATE, TIME, or TIMESTAMP, then let B_1 and E_1 both be the value of PD_1.
   (b) If the data type of PD_1 is PERIOD, then let B_1 be the value of BEGIN(PD_1) and E_1 be the value of END(PD_1) + INTERVAL '1' P.

(c) Otherwise, let R_1 be the value of <row value expression 1> and let B_1 be the value of the first column of R_1.

Case:

i) If the data type of the second column of R_1 is a datetime data type, then let E_1 be the value of the second column of R_1.

ii) If the data type of the second column of R_1 is INTERVAL, then let I_1 be the value of the second column of R_1. Let $E_1 = B_1 + I_1$.

2. If E_1 is the null value or if $E_1 < B_1$, then let $T_1 = B_1$. Otherwise, let $T_1 = E_1$.

3. Case:

(a) If the data type of PD_2 is DATE, TIME, or TIMESTAMP, then let B_2 and E_2 both be the value of PD_2.

(b) If the data type of PD_2 is PERIOD, then let B_2 be the value of BEGIN(PD_2) and E_2 be the value of END(PD_2) + INTERVAL '1' P.

(c) Otherwise, let R_2 be the value of <row value expression 2> and let B_2 be the value of the first column of R_2.

Case:

i) If the data type of the second column of R_2 is a datetime data type, then let E_2 be the value of the second column of R_2.

ii) If the data type of the second column of R_2 is INTERVAL, then let I_2 be the value of the second column of R_2. Let $E_2 = B_2 + I_2$.

4. If B_2 is the null value or if $E_2 < B_2$, then let $S_2 = E_2$. Otherwise, let $S_2 = B_2$.

5. The value of the <precedes predicate> is the value of $T_1 < S_2$.

A.3 <contains predicate>

Function

Specify a test for a period containing another period or a datetime.

Format

<contains predicate> ::=
    <period or datetime value expression 1> `CONTAINS` <period or datetime value expression 2>

Syntax Rules

1. Let <period or datetime value expression 1> be PD_1 and let <period or datetime value expression 2> be PD_2.
2. If PD_1 is <row value expression>, then:
   a) Let R_1 be the <row value expression> in PD_1.
   b) The degree of R_1 shall be 2.
   c) Let T_1 be the type of the first column of R_1, and T_2 be the type of the second column of R_1. T_1 shall be a datetime data type.
   d) If T_2 is a datetime data type, then it shall be comparable with T_1.
   e) If T_2 is INTERVAL, then the precision of T_2 shall be such that the interval can be added to T_1.
   f) The precision of PD_1 is the precision of T_1.
3. If PD_1 is <period value expression>, then the precision of PD_1 is that of the <period value expression>.
4. PD_1 shall not be a <datetime value expression>.
5. If PD_2 is <row value expression>, then:
   a) Let R_2 be the <row value expression> in PD_2.
   b) The degree of R_2 shall be 2.
   c) Let T_3 be the type of the first column of R_2, and T_4 be the type of the second column of R_2. T_3 shall be a datetime data type.
   d) If T_4 is a datetime data type, then it shall be comparable with T_3.
   e) If T_4 is INTERVAL, then the precision of T_4 shall be such that the interval can be added to T_3.
   f) The precision of PD_2 is the precision of T_3.
6. If PD_2 is <period value expression> or <datetime value expression>, then the precision of PD_2 is that of the <period value expression> or the <datetime value expression>.
7. The precision of PD_1 and PD_2 shall be identical. Let this precision be P.

Access Rules

No additional Access Rules.

General Rules

1. Case:
   (a) If the data type of PD_1 is PERIOD, then let B_1 be the value of BEGIN(PD_1) and E_1 be the value of END(PD_1) + INTERVAL '1' P.
   (b) Otherwise, let R_1 be the value of <row value expression 1> and let B_1 be the value of the first column of R_1.
   Case:

i) If the data type of the second column of R_1 is a datetime data type, then let E_1 be the value of the second column of R_1.

ii) If the data type of the second column of R_1 is INTERVAL, then let I_1 be the value of the second column of R_1. Let $E_1 = B_1 + I_1$.

2. If B_1 is the null value or if $E_1 < B_1$, then let $S_1 = E_1$ and let $T_1 = B_1$. Otherwise, let $S_1 = B_1$ and let $T_1 = E_1$.

3. Case:

   (a) If the data type of PD_2 is DATE, TIME, or TIMESTAMP, then let B_2 and E_2 both be the value of PD_2.

   (b) If the data type of PD_2 is PERIOD, then let B_2 be the value of BEGIN(PD_2) and E_2 be the value of END(PD_2) + INTERVAL '1' P.

   (c) Otherwise, let R_2 be the value of <row value expression 2> and let B_2 be the value of the first column of R_2.

   Case:

   i) If the data type of the second column of R_2 is a datetime data type, then let E_2 be the value of the second column of R_2.

   ii) If the data type of the second column of R_2 is INTERVAL, then let I_2 be the value of the second column of R_2. Let $E_2 = B_2 + I_2$.

4. If B_2 is the null value or if $E_2 < B_2$, then let $S_2 = E_2$ and let $T_2 = B_2$. Otherwise, let $S_2 = B_2$ and let $T_2 = E_2$.

5. The value of the <contains predicate> is the result of the following expression:

   $S_1 \leq S_2$ AND $T_2 \leq T_1$

A.4 <meets predicate>

Function

Specify a test for one period or datetime adjoining another period or datetime.

Format

<meets predicate> ::=
    <period or datetime value expression 1> `MEETS` <period or datetime value expression 2>

Syntax Rules

1. Let <period or datetime value expression 1> be PD_1 and let <period or datetime value expression 2> be PD_2.
2. If PD_1 is <row value expression>, then:
   a) Let R_1 be the <row value expression> in PD_1.
   b) The degree of R_1 shall be 2.
   c) Let T_1 be the type of the first column of R_1, and T_2 be the type of the second column of R_1. T_1 shall be a datetime data type.
   d) If T_2 is a datetime data type, then it shall be comparable with T_1.
   e) If T_2 is INTERVAL, then the precision of T_2 shall be such that the interval can be added to T_1.
   f) The precision of PD_1 is the precision of T_1.
3. If PD_1 is <period value expression> or <datetime value expression>, then the precision of PD_1 is that of the <period value expression> or the <datetime value expression>.
4. If PD_2 is <row value expression>, then:
   a) Let R_2 be the <row value expression> in PD_2.
   b) The degree of R_2 shall be 2.
   c) Let T_3 be the type of the first column of R_2, and T_4 be the type of the second column of R_2. T_3 shall be a datetime data type.
   d) If T_4 is a datetime data type, then it shall be comparable with T_3.
   e) If T_4 is INTERVAL, then the precision of T_4 shall be such that the interval can be added to T_3.
   f) The precision of PD_2 is the precision of T_3.
5. If PD_2 is <period value expression> or <datetime value expression>, then the precision of PD_2 is that of the <period value expression> or the <datetime value expression>.
6. The precision of PD_1 and PD_2 shall be identical. Let this precision be P.

Access Rules

No additional Access Rules.

General Rules

1. Case:
   (a) If the data type of PD_1 is DATE, TIME, or TIMESTAMP, then let B_1 and E_1 both be the value of PD_1.
   (b) If the data type of PD_1 is PERIOD, then let B_1 be the value of BEGIN(PD_1) and E_1 be the value of END(PD_1) + INTERVAL '1' P.

(c) Otherwise, let R_1 be the value of <row value expression 1> and let B_1 be the value of the first column of R_1.

Case:

i) If the data type of the second column of R_1 is a datetime data type, then let E_1 be the value of the second column of R_1.

ii) If the data type of the second column of R_1 is INTERVAL, then let I_1 be the value of the second column of R_1. Let $E_1 = B_1 + I_1$.

2. If B_1 is the null value or if $E_1 < B_1$, then let $S_1 = E_1$ and let $T_1 = B_1$. Otherwise, let $S_1 = B_1$ and let $T_1 = E_1$.

3. Case:

(a) If the data type of PD_2 is DATE, TIME, or TIMESTAMP, then let B_2 and E_2 both be the value of PD_2.

(b) If the data type of PD_2 is PERIOD, then let B_2 be the value of BEGIN(PD_2) and E_2 be the value of END(PD_2) + INTERVAL '1' P.

(c) Otherwise, let R_2 be the value of <row value expression 2> and let B_2 be the value of the first column of R_2.

Case:

i) If the data type of the second column of R_2 is a datetime data type, then let E_2 be the value of the second column of R_2.

ii) If the data type of the second column of R_2 is INTERVAL, then let I_2 be the value of the second column of R_2. Let $E_2 = B_2 + I_2$.

4. If B_2 is the null value or if $E_2 < B_2$, then let $S_2 = E_2$ and let $T_2 = B_2$. Otherwise, let $S_2 = B_2$ and let $T_2 = E_2$.

5. The value of the <meets predicate> is the result of the following expression:

T_1 + TIMESTAMP $10^{-P} = S_2$ OR T_2 + TIMESTAMP $10^{-P} = S_1$

B New Sections

The following sections should be added.

B.1 <interval value expression>

Function
Specify an interval value.
Format

<interval value expression> ::=
!! All alternatives from Part 2 of this International Standard
| <interval value function>

Syntax Rules
No additional Syntax Rules.
Access Rules
No additional Access Rules.
General Rules
No additional General Rules.

B.2 <interval value function>

Function
Specify a function yielding a value of type interval.
Format

<interval value function> ::=
`INTERVAL` <left paren> <period value expression> <right paren>

Syntax Rules

1. The data type of an <interval value function> is a day-time INTERVAL, of the same precision as the <period value expression>.

Access Fules
No additional Access Rules.
General Rules

1. Let PV be the value of <period value function> and let P be the precision of PV.
2. If PV is null, then the result of <period value function> is null. Otherwise, the value of <interval value function> is the result of:

   ((END(PV) - BEGIN(PV)) P) + INTERVAL '1' P