# Contents

# List of Tables

# Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardizatin. National bodies that are members of ISO or IEC participate in the development of International Standards through technical commitees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical commitees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of infroamtion technology, ISO and IEC have estrablished a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 9075, Part *V*, was prepared by Joint Technical Committee ISO/IEC JTC 1, Information technology.

ISO/IEC 9075 consists of the following parts, under the general title Information technology—Database languages—SQL:

— Part 1: Fundamentals and Basic Language

— Part *W*: SQL Global Transaction Management

— Part *X*: Call Level Interface

— Part *Y*: Persistent SQL Modules

— Part *Z*: Host Language Bindings.

# Introduction

The organization of this International Standard is as follows:

1) Clause 1, "Scope", specifies the scope of this part of this International Standard.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this part of this International Standard, constitute provisions of this part of this International Standard.

3) Clause 3, "Definitions, notations, and conventions", defines the notations and convensions used in this part of this International Standard.

4) Clause 4, "Concepts", presents concepts used in the support of valid time.

5) Clause 5, "Modified Language Syntax", provides syntactic additions to SQL-92 to support valid time.

In the text of this International Standard, Clauses begin a new odd-numbered page. Any resulting blank space is not significant.

# Information Technology—Database Languages—SQL Part *V*—SQL Temporal Support (SQL/Temporal)

# 1. Scope

This Part of International Standard ISO/IEC 9075 defines the facilities by which a conforming SQL-implementation can support temporal data.

**Note:** The framework for this International Standard is described by the Reference Model of Data Management (ISO/IEC 10032:1993).

# 2 Normative References

The following standards contain provisions that, through reference in this document, constitute provisions of this Part of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC qand ISO maintain registers of currently valid International Standards.

— ISO/IEC 9075:1992, *Information Technology—Database Languages—SQL.*

The language additions of this Part of this International Standard are fully upwardly compatible with SQL-92. Support for valid time has been added.

# 3 Definitions, notations, and conventions

This document adheres to the terminology defined in ISO/IEC 9075:1992.

## 3.1 Definitions

a) **valid time**: The valid time of a fact is the time when the fact is true in the modeled reality.

## 3.2 Notations

The syntax notation used in this Part of this International Standard is an extended version of BNF ("Backus Normal Form" or "Backus Naur Form").

This version of BNF is fully described in Part 1 of this International Standard.

The syntax description is a modification of the SQL-92 syntax description, and follows all conventions used therein.

## 3.3 Conventions

The convensions used in this Part of this International Standard are identical to those described in Part 1 of this International Standard.

# 4 Concepts

Here we briefly outline the major concepts behind the language extension. Much more discussion may be found in a separate commentary.

## 4.1 Period Data Type

SQL-92's datetime and interval data types are augmented with a *period* data time, of specifiable precision. New operators on periods are defined.

## 4.2 Time-lines

This proposal adds the valid time line. The valid time of a fact is the time that it was valid in reality.

## 4.3 Aggregates

The conventional SQL-92 aggregates are extended to apply over temporal domains. They are also extended to return time-varying values.

## 4.4 Valid-time Tables

The snapshot tables currently supported by SQL-92 continue to be available. *State* tables may also be specified. In such tables, each tuple is timestamped with a *temporal element*, which is a union of periods. As an example, the Employee table with attributes Name, Salary and Manager could contain the tuple (Tony, 10000, LeeAnn). The temporal element timestamp would record the maximal (noncontiguous) periods in which Tony made $10000 and had LeeAnn as his manager. Information about other values of Tony's salary or other managers would be stored in other tuples. The timestamp is implicitly associated with each tuple; it is not another column in the table. The precision of the timestamps within the temporal element can be specified.

Temporal elements are closed under union, difference, and intersection. Timestamping tuples with temporal elements is conceptually appealing and can support multiple representational data models. Dependency theory can be extended to apply in full to this temporal data model.

*Event* tables may also be specified. In such tables, each tuple is timestamped with an *instant set*. As an example, a Hired table with attributes Name and Position could contain the tuple (LeeAnn, Manager). The instant set timestamp would record the instant(s) when LeeAnn was hired as a Manager. Information about other values of her positions would be stored in other tuples. In this case, the timestamp is implicitly associated with each tuple.

In summary, there are three kinds of tables: snapshot (no temporal support beyond time columns), valid-time state tables (consisting of sets of tuples timestamped with valid-time elements), and valid-time event tables (timestamped with valid-time instant sets).

## 4.5 Schema Specification

The `CREATE TABLE` and `ALTER` statements are extended to allow specification of the valid-time aspects of temporal tables. The scale and precision of the valid timestamps can also be specified and later altered.

## 4.6 Restructuring

The `FROM` clause is extended to allow tables to be *restructured* so that the temporal elements associated with tuples with identical values on a subset of the columns are coalesced. For example, to determine when Tony made a Salary of $10000, independent of who his manager was, the Employee table could be restructured on the Name and Salary columns. The timestamp of this restructured tuple would specify the periods when Tony made $10000, information which might be gathered from several underlying tuples specifying different managers.

Similarly, to determine when Tony had LeeAnn as his manager, independent of his salary, the table would be restructured on the Name and Manager columns. To determine when Tony was an employee, independent of how much he made or who his manager was, the table could be restructured on only the Name column.

Restructuring can also involve *partitioning* of the temporal element or instant set into its constituent maximal periods or instants, respectively. Many queries refer to a continuous property, in which maximal periods are relevant.

## 4.7 Temporal Selection

The valid-time timestamp of a table may participate in predicates in the `WHERE` clause via `VALID()` applied to the table (or correlation variable) name. The operators have been extended to take temporal elements and instant sets as arguments.

## 4.8 Temporal Projection

Conventional snapshot tables, as well as valid-time tables, can be derived from underlying snapshot or valid-time tables. An optional `VALID` or `VALID INTERSECT` clause is used to specify the timestamp of the derived tuple.

## 4.9 Update

The update statements have been extended in a manner similar to the `SELECT` statement, to specify the temporal extent of the update.

## 4.10 Cursors

Cursors have been extended to optionally return the valid time of the retrieved tuple.

## 4.11 System Tables

The `TABLES` base table has been extended to include information on the valid time component (if present) of a table.

# 5 Modified Language Syntax

The organization of this section follows that of the SQL-92 document. The syntax is listed under corresponding section numbers in the SQL-92 document. All new or modified syntax rules are marked with a bullet ("•") on the left side of the production.

Where appropriate, we provide disambiguating rules to describe additional syntactic and semantic restrictions. We assume that the reader is familiar with the SQL-92 proposal, and that a copy of the proposal is available for reference.

# 6 Section 5 Lexical Elements

## 6.1 Section 5.2 <token> and <separator>

The production for the non-terminal <delimiter token> is augmented.

<delimiter token> ::=
- | <period string>

The production for the non-terminal <reserved word> is modified to add 9 reserved words. To conserve space, we do not copy the existing 227 reserved word definitions from the SQL-92 document.

<reserved word> ::=
- | `CONTAINS`
- | `EVENT`
- | `INSTANT`
- | `MEETS`
- | `PERIOD` | `PRECEDES`
- | `SNAPSHOT` | `STATE`
- | `VALID`

## 6.2 Section 5.3 <literal>

The production for the non-terminal <general literal> is augmented.

<general literal> ::=
- | <period literal>

<period literal> ::=
- `PERIOD` <period string>

<period string> ::=
- <quote> <left bracket> <datetime literal> <space> <minus sign> <space> <datetime literal> <right bracket>

Additional syntax rules:

1. A <period string> is any sequence of characters not containing a single <quote>.
2. Let $A$ and $B$ be valid <datetime string>s, representing datetimes $C$ and $D$. If the <period string> is identical to a '`[`' followed by $A$ followed by ' `-` ' followed by $B$ followed by '`]`', then the value of the <period string> is the period from $C$ to $D$.
3. The data type of a <period literal> is `PERIOD`.

Additional general rules:

1. Period literals are interpreted as follows. The beginning granule of the period is the first granule contained in the period, and the ending granule of the period is the last granule contained in the period.
2. If a datetime, period, or interval literal is incorrectly specified then an exception condition is raised: *data exception—invalid time value literal.*

# 7 Section 6 Scalar Expressions

## 7.1 Section 6.1 <data type>

The production for the non-terminal <data type> adds one new type.

<data type> ::=
- | <period type>

<period type> ::=
- `PERIOD` [ <period qualifier> ] [ `WITH TIME ZONE` ]

<period qualifier> ::=
- <left paren> <timestamp precision> <right paren>

Additional general rules:

1. The delimiting datetimes of a period shall have the same precision and scale.

## 7.2 Section 6.3 <table reference>

The production for the non-terminal <table reference> is replaced with the following. The first component can be more complex than a single <table name>, and multiple space-separated <correlation name>s are permitted.

<table reference> ::=
- <table source> [ [ `AS` ] <corr> { <corr> }... ]
- | <derived table> [ `AS` ] <corr> { <corr> }...
  | <joined table>

<corr> ::=
- <correlation>
- | <joined table>

The following productions are added. The first allows table references to be defined in terms of other table references. The rest serve to define <correlation modifier>.

<table source> ::=
- <table name> <correlation modifier>
- | <correlation name> <correlation modifier>

<correlation> ::=
- <correlation name> [ <left paren> <derived column list> <right paren> ]

<correlation modifier> ::=
- [ <left paren> <coalescing columns> <right paren> ]
  [ <left paren> <partitioning unit> <right paren> ]

<coalescing columns> ::=
- <column name> [ {<comma> <column name> }... ]
- | <asterisk>

<partitioning unit> ::=

- | INSTANT
- | PERIOD

Additional syntax rules:

1. <coalescing columns> of <asterisk> imply all the attributes of the <table name> or <correlation name>.
2. If the <coalescing attributes> are not present, then <asterisk> is assumed.
3. If a <correlation modifier> is applied to a <table source>, then a <correlation> is required.
4. If the <correlation modifier> is applied to a <correlation name>, then the attributes are drawn from the table upon which the <correlation name> is based, and augment those attributes associated with the <correlation name>. The latter attributes can be mentioned in this <correlation modifier>, but is not required.
5. If <partitioning unit> is not specified, then no partitioning is assumed.
6. Partitioning on `INSTANT` is only permitted for event tables.
7. Partitioning on `PERIOD` is only permitted for state tables.

Additional general rules:

1. Let $CM$ be the <correlation modifier>. Let $CN$ be a <column name> contained in $CM$, and $C$ be the column.

   Case:

   - If $CM$ is associated with a <table name>, then let $T$ be that table name. The table identified by $T$ is the *ultimate table* of $CN$.
   - If $CN$ is associated with a <correlation name>, then let $D$ be that <correlation name>. The ultmate table of $CN$ is the ultimate table of $D$.
2. $C$ must be a column of its ultimate table.
3. Only those <column name>s indicated as <coalescing columns> are accessible via the <correlation name>.

## 7.3 Section 6.5 <set function specification>

Additional syntax rules:

1. Let $DT$ be the data type of the <value expression>.
2. If `SUM` is specified, DT shall not be an instant or a period.
3. If `AVG` is specified, DT shall not be a period, an event set, or a temporal element.

Additional general rules:

1. If `MIN`, `MAX`, `SUM`, or `AVG` is specified and T is a timestamp, then

   Case:

   (a) If `MIN` is present, then use `PRECEDE` to determine the minimum timestamp, except in the case that A is an interval, in which case return the interval with the minimal number of granules.

(b) If `MAX` is present, then use not `PRECEDE` to determine the maximum timestamp, except in the case that A is an interval, in which case return the interval with the maximal number of granules.

(c) If `SUM` is present, if the type of A is an interval, then return an interval equal in length to the sum of the granules in T. Otherwise, the type of A must be a temporal element or event set, and the result is the result of set union of the elements of T.

(d) If `AVG` is present, if the type of A is an interval, then return an interval equal in length to the average number of granules in T. Otherwise, the type of A must be an instant. Pick any origin $O$. Compute the average of the distance from $O$ to each instant in T, and return the instant representing the distance from $O$ to this average.

2. If `SUM` is specified, $T$ is `INTERVAL` and the sum is not within the range of data type then an exception condition is raised: *data exception—time value out of range.*

## 7.4 Section 6.8 <datetime value function>

Expressions evaluating to or taking as a parameter periods or temporal expressions are added.

<datetime value function> ::=

- | `BEGIN` <left paren> <period value expression> <right paren>
- | `END` <left paren> <period value expression> <right paren>
- | `FIRST` <left paren> <datetime value expression> <comma> <datetime value expression> <right paren>
- | `LAST` <left paren> <datetime value expression> <comma> <datetime value expression> <right paren>
- | `FIRST` <left paren> <instant set value expression> <right paren>
- | `LAST` <left paren> <instant set value expression> <right paren>
- | `VALID` <left paren> { <table name> | <correlation name> } <right paren>

Additional general rules:

1. `FIRST` (`LAST`) extracts the first (last) instant from the instant set.

2. Use of `VALID` must be on valid event tables which are partitioned.

## 7.5 Section 6.10 <cast specification>

Casting between data types is extended to include the temporal types. No syntactic changes or additions are required to do this.

Additional syntax rules:

1. The table showing allowable data conversions is augmented to add the `PERIOD` (P), temporal element (TE) and instant set (IS) data types.

| <data type> of *SD* \ <data type> of *TD* | EN | AN | VC | FC | VB | FB | D | T | TS | YM | DT | P | TE | IS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| EN | Y | Y | Y | Y | N | N | N | N | N | M | M | N | N | N |
| AN | Y | Y | Y | Y | N | N | N | N | N | N | N | N | N | N |
| C | Y | Y | M | M | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| B | N | N | Y | Y | Y | Y | N | N | N | N | N | N | N | N |
| D | N | N | Y | Y | N | N | Y | N | Y | N | N | Y | Y | Y |
| T | N | N | Y | Y | N | N | N | Y | Y | N | N | Y | Y | Y |
| TS | N | N | Y | Y | N | N | Y | Y | Y | N | N | Y | Y | Y |
| YM | M | Y | Y | Y | N | N | N | N | N | Y | N | N | N | N |
| DT | M | Y | Y | Y | N | N | N | N | N | N | Y | N | N | N |
| P | N | N | Y | Y | N | N | N | N | N | M | M | Y | Y | N |
| TE | N | N | N | N | N | N | N | N | N | N | N | N | Y | Y |
| IS | N | N | N | N | N | N | N | N | N | N | N | N | Y | Y |

2. If *SD* is D, T, or TS and *TD* is P then the conversion results in a period of duration one granule.
3. If *SD* is C and *TD* is P then the conversion is first done to the T data type.
4. If *SD* is C, D, T, or TS and *TD* is TE then the conversion is first done to the *P* type.
5. If *SD* is P and *TD* is TE, then the conversion is into a temporal element containing one period.
6. If *SD* is C, T, TS, or P and *TD* is IS then the conversion is first done to the *TE* type.
7. If *SD* is TE and *TD* is IS then the conversion is done by applying `FIRST` to each period in the set.

## 7.6 Section 6.11 <value expression>

The production for the non-terminal <value expression> is augmented to include expressions evaluating to periods, to temporal elements, and to instant sets.

<value expression> ::=
- | <period value expression>
- | <temporal element value expression>
- | <instant set value expression>

## 7.7 Section 6.14 <datetime value expression>

Table 1 has been added, and includes periods.

## 7.8 Section 6.?? <period value expression>

This is a new section.

<period value expression> ::=
- <period primary>
- | <interval value expression> <plus sign> <period value expression>
- | <period value expression> {<plus sign> | <minus sign>} <interval value expression>

| *Operand 1* | *Operator* | *Operand 2* | *Yields* |
|---|---|---|---|
| | - | *interval* | *interval* |
| *interval* | + | *interval* | *interval* |
| *interval* | - | *interval* | *interval* |
| *datetime* | + | *interval* | *datetime* |
| *datetime* | - | *interval* | *datetime* |
| *interval* | + | *datetime* | *datetime* |
| *datetime* | - | *datetime* | *interval* |
| *interval* | * | *numeric* | *interval* |
| *numeric* | * | *interval* | *interval* |
| *interval* | / | *numeric* | *interval* |
| *interval* | / | *interval* | *numeric* |
| *interval* | + | *period* | *period* |
| *period* | + | *interval* | *period* |
| *period* | - | *interval* | *period* |

Table 1: Permitted Arithmetic Expressions and Results.

<period primary> ::=
- <period literal>
- | <column reference>
- | <scalar subquery>
- | <case expression>
- | <period value function>
- | <cast specification>

Additional syntax rules:

1. The data type of a <period value expression> is `PERIOD`.
2. Table 1 lists the arithmetic expressions involving time that are permitted.

Additional general rules:

1. If a temporal arithmetic operation yields a `PERIOD` value that is out of range then an exception condition is raised: *data exception—time value out of range.*

## 7.9 Section 6.?? <period value function>

This is a new section.

<period value function> ::=
- `VALID` <left paren> { <table name> | <correlation name> } <right paren>
- | `PERIOD` <left paren> <datetime value expression> <comma> <datetime value expression> <right paren>
- | `INTERSECT` <left paren> <period value expression> <comma> <period value expression> <right paren>
- | `FIRST` <left paren> <temporal element value expression> <right paren>
- | `LAST` <left paren> <temporal element value expression> <right paren>

Additional general rules:

1. Use of `VALID` is permitted only on valid time state tables that are partitioned, and denotes a maximal period in the timestamp of the underlying tuple.

2. `FIRST` (`LAST`) extracts the first (last) maximal period from the temporal element.

## 7.10 Section 6.?? <temporal element value expression>

The following are new nonterminals introduced into the language.

<temporal element value expression> ::=
- <temporal element value term>
- | <temporal element value expression> { <plus sign> | <minus sign> }
  <temporal element value term>

<temporal element value term> ::=
- <temporal element value factor>

<temporal element value factor> ::=
- <temporal element value primary>

<temporal element value primary> ::=
- <temporal element value function>

Additional general rules:

1. '`+`' ('`-`') on temporal elements is set union (difference).

## 7.11 Section 6.?? <temporal element value function>

A new nonterminal, <temporal element value function>, is added.

<temporal element value function> ::=
- `VALID` <left paren> { <table name> | <correlation name> } <right paren>
- | `INTERSECT` <left paren> <temporal element value expression> <comma>
  <temporal element value expression> <right paren>

Additional general rules:

1. Use of `VALID` denotes the temporal element timestamping of the underlying tuple, which must be associated with a valid time state table that has not been partitioned.

2. Intersection of temporal elements is set intersection.

## 7.12 Section 6.?? <instant set value expression>

The following are new nonterminals introduced into the language.

<instant set value expression> ::=
- <instant set value primary>
- | <instant set value expression> { <minus> | <plus> } <instant set value primary>

<instant set value primary> ::=
- <instant set value function>

Additional general rules:

1. '+' ('-') on instant sets is set union (difference).

## 7.13 Section 6.?? <instant set value function>

A new nonterminal, <instant set value function>, is added.

<instant set value function> ::=
- `VALID` <left paren> { <table name> | <correlation name> } <right paren>
- | `INTERSECT` <left paren> <instant set value expression> <comma>
  <instant set value expression> <right paren>

Additional general rules:

1. Use of `VALID` denotes the instant set timestamping of the underlying tuple, which must be associated with a valid-time event table that has not been partitioned.

# 8 Section 7 Query expressions

## 8.1 7.1 <row value constuctor>

A tuple can now include a valid time.

<row value constructor> ::=
<row value constructor element>
• | <left paren> <row value constructor list> <right paren> [ <valid value> ]
| <row subquery>

<valid value> ::=
• VALID { <element value expression> | <interval value expression>
| <event value expression> | <event set value expression> }

## 8.2 Section 7.3 <table expression>

The production for the non-terminal <table expression> is replaced with the following, adding one clause.

<table expression> ::=
• [ <valid clause> ]
<from clause>
[<where clause> ]
[ <group by clause>]
[ <having clause> ]

The following production is added.

<valid clause> ::=
• { VALID | VALID INTERSECT } { <temporal element value expression>
| <period value expression> | <datetime value expression>
| <instant set value expression> }

Additional general rules:

1. `VALID INTERSECT` $T$ is equivalent to

   `VALID INTERSECT`($T$, `INTERSECT`($C_1$, ..., `INTERSECT`($C_{n-1}$, $C_n$)))

   The correlation variables are listed in order of increasing precision (finer to coarser).

   where $C_i$ are the correlation variables (or table names) mentioned in the `SELECT` clause.

2. The default `VALID` clause is

   `VALID INTERSECT PERIOD '[0001/01/01 - 9999/12/31]'`.

3. If the `VALID` clause specifies a period or instant value, the values from the other value-equivalent tuples are gathered into a temporal element or instant set, respectively.

## 8.3 Section 7.9 <query specification>

The production is replaced with the following, adding one optional reserved word.

<query specification> ::=

- `SELECT` [ <set quantifier> ] [ `SNAPSHOT` ] <select list> <table expression>

Additional general rules:

1. `SNAPSHOT` specifies that the resulting table will be a snapshot table. In this case, the <table expression> should not include a <valid clause>.

# 9 Section 8 Predicates

## 9.1 Section 8.1 <predicate>

The production for the non-terminal <predicate> is replaced with the following.

<predicate> ::=
<comparison predicate>
| <between predicate>
| <in predicate>
| <like predicate>
| <null predicate>
| <quantified comparison predicate>
| <exists predicate>
| <unique predicate>
| <match predicate>
- | <precedes predicate>
- | <meets predicate>
- | <overlaps predicate>
- | <contains predicate>

## 9.2 Section 8.2 <comparison predicate>

No new syntax rules are required, but additional disambiguating rules are required for interval comparison.

1. The <less than operator>, <greater than operator>, and <equals operator> are valid for interval comparison.

## 9.3 Section 8.7 <quantified comparison predicate>

No additional productions are required. The following syntax rules are added.
Additional syntax rules:

1. Let $T_1$ be the type of <value expression>.
2. Let $T_2$ be the type of <row value expression>.
3. If either $T_1$ or $T_2$ is `DATE`, `TIME`, `TIMESTAMP`, `PERIOD` or `INTERVAL` then $T_1$ and $T_2$ must be comparable as defined in Table 2.

## 9.4 Section 8.11 <overlaps predicate>

The following productions are added for the new comparison operators. (The production for the `OVERLAPS` predicate is extended.) The applicable types are broadened to include temporal elements.

<overlaps predicate> ::=
<row value constructor 1> `OVERLAPS` <row value constructor 2>
- | <row value expression 1> `OVERLAPS` <row value expression 2>

<precedes predicate> ::=
- <row value expression 1> `PRECEDES` <row value expression 2>

<meets predicate> ::=
- <row value expression 1> `MEETS` <row value expression 2>

<contains predicate> ::=
- <row value expression 1> `CONTAINS` <row value expression 2>

This grammar is overly permissive in that it generates semantically illegal expressions. This is, however, consistent with the grammar originally provided in the SQL-92 standard for datetime value comparison. Expressions violating type constraints will be detected during semantic analysis.
Additional syntax rules:

1. Let $T_1$ be the type of <row value expression 1>.
2. Let $T_2$ be the type of <row value expression 2>.
3. $T_1$ and $T_2$ must be either `PERIOD` or *datetime*.
4. $T_1$ and $T_2$ shall be comparable as defined in Table 2.
5. Any comparison involving the `PERIOD` or *datetime* data types not listed in Table 2 is disallowed.

| *Operand 1* | *Operator* | *Operand 2* |
|---|---|---|
| *interval* | = | *interval* |
| *interval* | < | *interval* |
| *interval* | > | *interval* |
| *datetime/period/element* | = | *datetime/period/element* |
| *datetime/period/element* | `PRECEDES` | *datetime/period/element* |
| *datetime/period/element* | `OVERLAPS` | *datetime/period/element* |
| *datetime/period/element* | `CONTAINS` | *datetime/period/element* |
| *datetime/period/element* | `MEETS` | *datetime/period/element* |

Table 2: Permitted Set of Comparison Operators

# 10 Section 11 Schema definition and manipulation

## 10.1 Section 11.3 <table definition>

The production for the non-terminal <table definition> is augmented with an additional, optional clause.

<table definition> ::=
    CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE <table name>
      <table elements>
•       [ <temporal definition> ]
        [ ON COMMIT { DELETE | PRESERVE } ] ROWS ]

One production is added.

<temporal definition> ::=
•   AS { VALID [ STATE | EVENT ] } [ <timestamp precision> ]

Additional general rules:

1. Case:
   (a) If VALID is not specified, the table is a snapshot table.
   (b) If AS VALID STATE is specified, then the tuples are timestamped with valid-time elements that are sets of non-contiguous periods. The precision and scale of the periods can be specified.
   (c) If AS VALID EVENT is specified, then the tuples are timestamped with valid-time instant sets. The precision and scale of the instants can be specified.

## 10.2 Section 11.5 <default clause>

The production for the non-terminal <default clause> is changed to the following.

<default clause> ::=
      <literal>
    | <datetime value function>
•   | <interval value function>
•   | <period value function>
    | USER
    | SYSTEM USER
    | NULL

Additional syntax rules:

1. Let $T$ be the type of the column being initialized.
2. If $T$ is DATE, TIME, TIMESTAMP, PERIOD, or INTERVAL then USER and SYSTEM USER may not be specified.
3. If $T$ is DATE, TIME or TIMESTAMP then either a <literal> representing a <datetime literal> or a <datetime value function> may be specified.
4. If $T$ is PERIOD then either a <literal> representing a <period literal> or a <period value function> may be specified.
5. If $T$ is INTERVAL then either a <literal> representing an <interval literal> or an <interval value function> may be specified.

## 10.3 Section 11.10 <alter table statement>

The <alter table statement> is augmented with the following alternatives.

<alter table action> ::=
- | <add valid definition>
- | <drop valid definition>
- | <replace valid definition>

The following productions are added.

<add valid definition> ::=
- ADD [ VALID ] { STATE | EVENT } [ <timestamp precision> ]

<drop valid definition> ::=
- DROP VALID

<replace valid definition> ::=
- REPLACE [ VALID ] [ { STATE | EVENT } ] [ <timestamp precision> ]

Additional syntax rules:

1. Let T be the table identified in the containing <alter table statement>.
2. For the <add valid definition>, T shall be a snapshot or transaction-time table.
3. For the <drop valid definition>, T shall be a valid-time table.
4. For the <replace valid definition>, T shall be a valid-time table.

Additional general rules:

1. For the <drop valid definition>, if T is a state table, it is converted to a snapshot table with contents

   ```
   SELECT SNAPSHOT * FROM T WHERE T OVERLAPS CURRENT_TIMESTAMP
   ```

   If T is an event table, it is converted to a snapshot table with contents

   ```
   SELECT SNAPSHOT * FROM T
   ```

2. If $T$ was an state table and <valid definition> specifies period, then only the precision or scale of T's valid-time timestamps is altered. The temporal element of each tuple of $T$ is converted to the new precision and scale. If the scale in increased, the additional fractional digits are set to zero.
3. If $T$ was an state table and <valid definition> specifies event, then the timestamp of each tuple in $T$ is converted from a set of periods to a set of instants, equivalently,

   ```
   SELECT * VALID BEGIN(T) FROM T(PERIOD)
   ```

4. If $T$ was an event table and <valid definition> specifies event, then only the precision or scale of T's valid-time timestamps is altered. The instants in the timestamp of each tuple of $T$ are converted to the new precision and scale. If the scale in increased, the additional fractional digits are set to zero.
5. If $T$ was an event table and <valid definition> specifies period, then the timestamp of each tuple in $T$ is converted from a set of instants to a set of periods, equivalently,

   ```
   SELECT * VALID PERIOD(T, T) FROM T(EVENT)
   ```

# 11 Section 13 Data manipulation

## 11.1 Section 13.3 <fetch statement>

<fetch statement> ::=
    `FETCH` [ [ <fetch orientation> ] `FROM` ] <cursor name> [ `INTO` <fetch target list> ]
•   [ `INTO VALID` [ `PERIOD` ] <fetch target list> ]

Additional syntax rules:

1. At least one of `INTO` <fetch target list> and `INTO VALID` [ `PERIOD` ] <fetch target list> must be present in a fetch statement.

Additional general rules:

1. When a <fetch target list> follows `INTO VALID PERIOD`, it must contain precisely a single <target specification>. When a <fetch target list> follows `INTO VALID` (without `PERIOD`), it must contain exactly two <target specification>s.
2. When a <fetch target list> follows `INTO VALID PERIOD`, it must contain precisely a single <target specification>. This is only allowed with a state table is being evaluated by the `SELECT` statement. When a <fetch target list> follows `INTO VALID` (without `PERIOD`), it must contain exactly two <target specification>s if a state table is being evaluated by the `SELECT` statement, and exactly one <target specfication> is an event table is being evaluated.

## 11.2 Section 13.5 <select statement: single row>

The production is replaced with the following, adding one optional reserved word.

<select statement: single row> ::=
•   `SELECT` [ <set quantifier> ] [ `SNAPSHOT` ] <select list>
      `INTO` <select target list>
        <table expression>

Additional general rules:

1. `SNAPSHOT` specifies that the resulting table will be a snapshot table. In this case, the <table expression> should not include a <valid clause>.

## 11.3 Section 13.7 <delete statement: searched>

The production for the non-terminal <delete statement: searched> is augmented with an additional, optional clause. This clause references the non-terminal <valid clause> defined for the SELECT statement.

<delete statement: searched> ::=
    `DELETE FROM` <table name>
      [ `WHERE` <search condition> ]
•       [ <valid value> ]

Additional general rules:

1. If $T$ is a valid-time table, and the <valid value> is omitted, then the default valid value specified in the <table definition> is assumed. If there was no default value specified, then the interval

   `PERIOD(TIMESTAMP CURRENT_TIMESTAMP, NOBIND(TIMESTAMP 'now'))`

   is assumed.

## 11.4 Section 13.10 <update statement: searched>

<update statement: searched> ::=
UPDATE <table name>
SET <set clause list>
• [ <valid value> ]
[ WHERE <search condition> ]

# 12 Section 21 Information Schema and Definition Schema

## 12.1 Section 21.3.8 TABLES base table

```
ALTER TABLE TABLES ADD COLUMN
       VALID_TIME        CHARACTER_DATA
             CONSTRAINT VALID_TIME_CHECK
                        CHECK (VALID_TIME IN ('STATE', 'EVENT', 'NONE'))
```

## 12.2 Section 22 Status Codes

The exception codes associated with the SQLSTATE parameter are modified to support the period data type

We show only the changed exceptions.

| Condition | Class | Subcondition | Subclass |
|---|---|---|---|
| data exception | 22 | time value out of range | 008 |
| | | invalid time value literal | 007 |