# TSQL2 Language Specification

Richard T. Snodgrass (chair)
Ilsoo Ahn   Gad Ariav   Don Batory   James Clifford
Curtis E. Dyreson   Ramez Elmasri   Fabio Grandi
Christian S. Jensen   Wolfgang Käfer   Nick Kline   Krishna Kulkarni
T. Y. Cliff Leung   Nikos Lorentzos   John F. Roddick
Arie Segev   Michael D. Soo   Suryanarayana M. Sripada

September, 1994

Correspondence may be directed to the chair of the TSQL2 Language Design Committee, Richard T. Snodgrass, Department of Computer Science, University of Arizona, Tucson, AZ 85721, rts@cs.arizona.edu. The affiliations and e-mail addresses of the TSQL2 Language Design Committee members may be found in a separate section at the end of the document.

# Contents

# 1 Introduction

This document specifies a temporal extension to the SQL-92 language standard. The language is designated **TSQL2**.

The document is organized as follows. The next section indicates the starting point of the design, the SQL-92 language. Section 4 lists the desired features on which the TSQL2 Language Design Committee reached consensus. Section 5 presents the major concepts underlying TSQL2. Compatibility with SQL-92 is the topic of Section 6. Section 7 briefly discusses how the language can be implemented. Subsequent sections specify the syntax of the language extensions.

# 2 Normative References

The following standards contain provisions that, through reference in this document, constitute provisions of this language specification.

— ISO/IEC 9075:1992, *International Organization for Standardization/International Electrotechnical Commission—Database Language SQL.*

TSQL2 is a fully upwardly compatible extension of SQL-92. The functionality of user-defined time support in SQL-92 is enhanced. Support for valid and transaction time has been added.

The language has been evaluated against the Test Suite of Temporal Database Queries.

# 3 Definitions, notations, and conventions

This document adheres to the terminology defined in ISO/IEC 9075:1992. Where possible, the document also adheres to the terminology defined in the consensus temporal database glossary.

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

# 4 Desired Features

This section lists the desired features that should be supported by TSQL2. These features guided the design of the language.

We first considered aspects of the data model.

- TSQL2 should not distinguish between snapshot equivalent instances, i.e., snapshot equivalence and identity should be synonymous.

  This provides conceptual simplicity.

- TSQL2 should support only one valid-time dimension.

- For simplicity, tuple timestamping should be employed.

- TSQL2 should be based on homogeneous tuples.

- Valid time support should include support for both the past and the future.

  While some existing temporal models only include valid time support up to now, it is important to provide support for future valid time so that planning activities can be accommodated.

- Timestamp values should not be limited in range or precision.

  SQL-92 is limited to A.D., to 9999 years, and to an excessive coarse precision of seconds for a representation of 20 positions. It is also not sufficiently defined (e.g., addition is implementation defined!) For temporal databases to be used in scientific applications, as well as by historians and others requiring an extended range, the representation and semantics must be extended and be better defined.

We then considered the language proper.

- TSQL2 should be a consistent, fully upwardly compatible extension of SQL-92.
- TSQL2 should allow the restructuring of tables on any set of attributes.

  Such an ability was first proposed in the TempSQL language proposal.
- TSQL2 should allow for flexible temporal projection, but TSQL2 syntax should reveal clearly when non-standard temporal projections are being done.
- Operations in TSQL2 should not accord any explicit attributes special semantics.

  For example, operations should not rely on the notion of a key.
- Temporal support should be optional, on a per-table basis.

  Tables that are not specified to be temporal should be considered to be snapshot tables. It is important to be an extension of SQL-92's data model when possible, not a replacement. Hence, the schema definition language should allow the definition of snapshot tables, when temporal support is not desired. Similarly, it should be possible to derive a snapshot table from a temporal table.
- User-defined time support should include instants, periods, and intervals.

  User-defined time support in SQL-92 is greatly flawed. C.J. Date has listed many of the problems with it.
- Existing aggregates should have temporal analogues in TSQL2.

  It is important that existing language features such as aggregates still apply in the temporal data model.
- Multiple calendar and multiple language support should be present in timestamp input and output, and timestamp operations.

  SQL-92 supports only one calendar, a particular variant of the Gregorian calendar, and one time format. The many uses of temporal databases demand much more flexibility.
- It should be possible to derive temporal and non-temporal tables from underlying temporal and non-temporal tables.

Finally, we made ease of implementation a priority.

- TSQL2 tables should be implementable in terms of tables in some first normal form representational model.

  In particular, the language should be implementable via a data model that employs period-timestamped tuples. This is the most straightforward representational model, in terms of extending current relational technology. Nevertheless, the language should *accept* implementation using other representational models, such as attribute timestamped representational models.
- TSQL2 must have an efficiently implementable algebra that allows for optimization and that is an extension of the snapshot algebra.

  Current DBMS implementations are based on the snapshot algebra. The temporal algebra used with the TSQL2 temporal data model should contain temporal operators that are extensions of the operations in the snapshot algebra. Snapshot reducibility is also highly desired, so that, for example, optimization strategies will continue to work in the new data model.
- The language data model should allow multiple representational data models.

  In particular, it would be best if the data model accommodated the major temporal data models proposed to date, including attribute timestamped models.

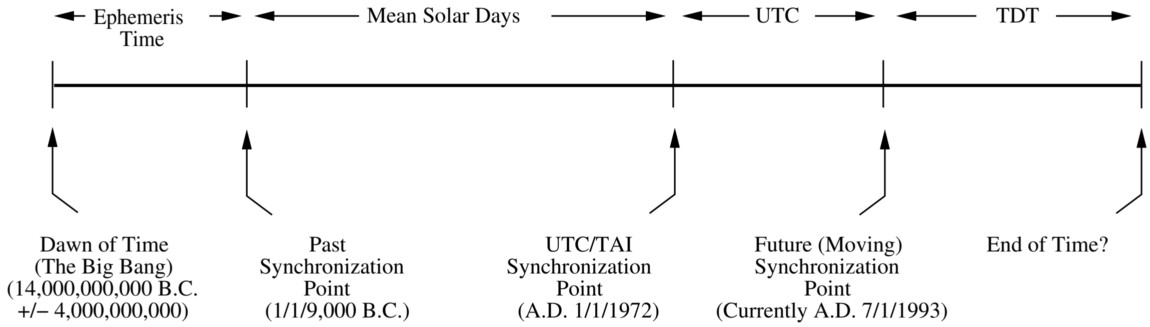


Figure 1: The Base Line Clock

# 5 Concepts

Here we briefly outline the major concepts behind the TSQL2 extension. Much more discussion may be found in the commentaries.

## 5.1 Time Ontology

The TSQL2 model of time is bounded on both ends. The model refrains from deciding whether time is ultimately continuous, dense, or discrete. TSQL2 does not allow the user to ask a question that will differentiate the alternatives. Instead, the model accommodates all three alternatives by assuming that an instant on a time-line is much smaller than a chronon, which is the smallest entity that a timestamp can represent exactly (the size of a chronon is implementation-dependent). An instant can only be approximately represented. A discrete image of the represented times emerges at run-time as timestamps are scaled to user-specified (or default) granularities and as operations on those timestamps are performed to the given scale.

An instant is modeled by a timestamp coupled with an associated scale (e.g., day, year, month). A period is modeled by the composition of two instant timestamps and the constraint that the instant timestamp that starts the period equals or precedes (in the given scale) the instant timestamp that terminates the period.

## 5.2 Base Line Clock

A semantics must be given to each time that is stored in the database. SQL-92 specifies that times are given in UTC seconds, which are not defined before 1958, and in any case cannot be used to date prehistoric time, as UTC is based in part on solar time.

TSQL2 includes the concept of a *baseline clock*, which provides the semantics of timestamps. The baseline clock relates each second to physical phenomena. Since we are targeting use for a general-purpose database, we attempted to anticipate the needs of an average database user and to provide a baseline clock that meets those needs.

The baseline clock is shown in Figure 1 (not to scale). It partitions the time line into a set of contiguous *periods*. Each period runs on a different clock. A *synchronization point* delimits a period boundary. The baseline clock and its representation are independent of any calendar. We use Gregorian calendar dates in this discussion only to provide an informal indication of when the synchronization points occur.

From the Big Bang to Midnight January 1, 9000 B.C. the baseline clock runs on ephemeris time. For historic instants, 9000 B.C. to January 1, 1972, the baseline clock follows the mean solar day clock. The mean solar clock carries the baseline clock up to Midnight January 1, 1972 after which the baseline clock follows UTC. Midnight January 1, 1972 is when UTC was synchronized with the atomic clock and the current system of leap seconds was adopted. In particular, during the period in which the baseline clock uses UTC, 26 leap seconds were added. The baseline clock runs on UTC until one second before Midnight, January 1, 1995. This is the next time at which a leap second might be added (leap second announcements are made by the International Earth Rotation Service). After Midnight January 1, 1995, until the "Big Crunch" or the end of our baseline clock, the baseline clock follows Terrestrial Dynamic Time (TDT) since both UTC and mean solar time are unknown and unpredictable. Also, since 1984, TDT has been favored over ephemeris time by the international standards community.

### 5.3 Data Types

SQL-92's datetime and interval data types are augmented with a *period* data time, of specifiable range and precision. The range and precision can be expressed as an integer (e.g., a precision of 3 fractional digits) or as an interval (e.g., a precision of a week). Operators are available to compare timestamps and to compute new timestamps, with a user-specified precision. Temporal values can be input and output in user-specifiable formats, in a variety of natural languages. *Calendars* and *calendric systems* permit the application-dependent semantics of time to be incorporated.

A surrogate data is introduced in TSQL2. Surrogates are unique identifiers that can be compared for equality, but the values of which cannot be seen by the users. In this sense, a surrogate is "pure" identity and does not describe a property (i.e., it has no observable value). Surrogates are useful in identifying objects associated with time-varying attributes, but are not a replacement for keys.

### 5.4 Time-lines

Three time-lines are supported in TSQL2: user-defined time, valid time, and transaction time. All three have the ontology described above. Hence values from disparate time-lines can be compared, at an appropriate precision. Transaction-time is bounded by `initiation`, the time when the database was created, and `until changed`. In addition, user-defined and valid time have two special values, `beginning` and `forever`, where are the least and greatest values in the ordering. Transaction time has the special value `until changed`.

Valid and user-defined data types can be *temporally indeterminate*. In temporal indeterminacy, it is known that an event stored in a temporal database did in fact occur, but it is not known exactly *when* that event occurred. An instant (interval, period) can be specified as determinate or indeterminate; if the latter, then the possible mass functions, as well as the generality of the indeterminacy to be represented can be specified. The quality of the underlying data (termed its *credibility*) and the *plausibility* of the ordering predicates expressed in the query can be controlled on a per-query or global basis.

Finally, temporal values (instant timestamps) can be *now-relative*. A now-relative time of "now - 1 day", interpreted when the query was executed on June 12, 1993, would have the *bound* value of "June 11, 1993." The user can specify whether values to be stored in the database are to be bound (i.e., not now-relative) or unbound.

### 5.5 Aggregates

The conventional SQL-92 aggregates are extended to apply over temporal domains. They are also extended to return time-varying values and to permit grouping via a partitioning of the underlying time line, termed *temporal grouping*. Values can be *weighted* by their duration during the computation of the aggregate. Finally, one new temporal aggregate, `RISING` is added. A taxonomy of temporal aggregates identifies fourteen possible kinds of aggregates; there are instances of all of these kinds in TSQL2.

### 5.6 Valid-time Tables

The snapshot tables currently supported by SQL-92 continue to be available in TSQL2. TSQL2 also allows *state* tables to 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 range, precision and indeterminacy 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.

TSQL2 also allows *event* tables to 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 tuples.

## 5.7 Transaction-time and Bitemporal Tables

Orthogonally to valid time, transaction time can be associated with tables. The transaction time of a tuple, which is a temporal element, specifies when that tuple was considered to be logically stored in the database. If the tuple (Tony, 10000, LeeAnn) was stored in the database on March 15, 1992 (say, with an `APPEND` statement) and removed from the database on June 1, 1992 (say, with a `DELETE` statement), then the transaction time of that tuple would be the period from March 15, 1992 to June 1, 1992.

The transaction timestamps have an implementation-dependent range and precision, and are determinate.

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

## 5.8 Schema Specification

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

## 5.9 Restructuring

The `FROM` clause in TSQL2 allows 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.

### 5.10 Temporal Selection

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

### 5.11 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. The transaction time of an appended or modified tuple is supplied by the DBMS.

### 5.12 Update

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

### 5.13 Cursors

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

### 5.14 Schema Versioning

Schema *evolution*, where the schema may change, is already supported in SQL92. However, old schemas are discarded; the data is always consistent with the current schema. Transaction time support dictates that previous schemas be accessible, termed *schema versioning*. TSQL2 supports a minimal level of schema versioning.

### 5.15 Vacuuming

Updates, including (*logical*) deletions, to transaction time tables result in insertions at the physical level. Despite the continuing decrease in cost of data storage, it is still, for various reasons, not always acceptable that all data be retained forever. TSQL2 supports a simple form of *vacuuming*, i.e., physical deletion, of data when such tables are being managed.

### 5.16 System Tables

The `TABLES` base table has been extended to include information on the valid and transaction time components (if present) of a table. Two other base tables have been added to the definition schema.

## 6 SQL-92 Compatibility

All aspects of TSQL2 are pure extensions of SQL-92. The user-defined time in TSQL2 is a consistent replacement for that of SQL-92. This was done to permit support of multiple calendars and literal representations. Legacy applications can be supported through a default `SQL92_calendric_system`.

The defaults for the new clauses used to support temporal tables were designed to satisfy snapshot reducibility, thereby ensuring that these extensions constitute a strict superset of SQL-92.

# 7 Implementation

During the design of the language, considerable effort was expended to ensure that the language could be implemented with only moderate modification to a conventional SQL-92-compliant DBMS. In particular, an algebra has been demonstrated that can be implemented in terms of a period-stamped (or instant-stamped, for event tables) tuple representational model; few extensions to the conventional algebra were required to fully support the TSQL2 constructs. This algebra is snapshot reducible to the conventional relational algebra.

Support for multiple calendars, multiple languages, mixed precision, and indeterminacy have been included in prototypes that demonstrated that these extensions have little deleterious effect on execution performance.

Mappings from the data model underlying TSQL2, the bitemporal conceptual data model, to various representational data models have been given elsewhere.

# 8 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.

# 9 Section 4 Concepts

## 9.1 Section 4.5 Datetimes and intervals

This section is replaced with the material found above in Section 5.

# 10 Section 5 Lexical Elements

## 10.1 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 25 reserved words. To conserve space, we do not copy the existing 227 reserved word definitions from the SQL-92 document.

<reserved word> ::=
- | `CALENDRIC` | `CONTAINS` | `CREDIBILITY`
- | `DISTRIBUTION`
- | `EVENT`
- | `GENERAL`
- | `INAPPLICABLE` | `INDETERMINATE`
- | `MEETS`
- | `NEW` | `NOBIND` | `NONSTANDARD`
- | `PERIOD` | `PLAUSIBILITY` | `PRECEDES` | `PREVIOUS` | `PROPERTIES`
- | `RISING`
- | `SCALE` | `SNAPSHOT` | `STATE` | `SURROGATE`
- | `VACUUM`
- | `VALID`
- | `WEIGHTED`

## 10.2 Section 5.3 <literal>

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

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

The <date string>, <time string>, <timestamp string>, and <interval string> are generalized. The <year-month literal>, <day-time literal>, <day-time interval>, <time interval>, <years value>, <months value>, <days value>, <hours value>, <minutes value>, <seconds value>, <seconds integer value>, <seconds fraction> and <datetime value> productions are all removed.

The allowable datetime, interval, and period literals is expanded to support multiple character sets, user-specified representations, and indeterminate and now-relative values.

<date literal> ::=
- `DATE` <date string> <calendric-property specification>

<time literal> ::=
- `TIME` <time string> <calendric-property specification>

<timestamp literal> ::=
- `TIMESTAMP` <timestamp string> [ <timestamp precision> ] <calendric-property specification>

<interval literal> ::=
- `INTERVAL` [ <sign> ] <interval string> <interval qualifier> <calendric-property specification>

<date string> ::=
- <datetime string>

<datetime string> ::=
- <character string literal>
- | <determinate datetime string>
- | <now-relative datetime string>
- | <indeterminate now-relative datetime string>
- | <now-relative with indeterminate datetime string>

<time string> ::=
- <datetime string>

<timestamp string> ::=
- <datetime string>

<interval string> ::=
- <character string literal>
- | <determinate interval string>
- | <indeterminate interval string>
- | <now-relative interval string>

<period literal> ::=
- `PERIOD` <period string> [ <period precision> ] <calendric-property specification>

<period string> ::=
- <character string literal>

Format-related property values describe the contents of temporal constants. The BNF grammar for format strings is as follows.

<format string> ::=
- <quote> [ <character representation> | field specification> ]... <quote>

<field specification> ::=
- <less than operator> <field identifier>
  [ <comma> <translation table name> [ <comma> <field formatting specification>...] ]
  <greater than operator>

<field identifier> ::=
- <identifier>

<translation table name> ::=
- <identifier>

<field formatting specification> ::=
- `W` <unsigned integer>
- `L`
- `R`
- `Z`
- `B`
- `S`

Syntax rules 7, 19, 20, 21, 22, 23 and 24 are removed, as they impose fairly arbitrary restrictions on timestamps.
Additional syntax rules:

1. A <format string> defines the syntax of strings specified as the values of format properties in property tables. A <format string> must be contained in an activated property table to affect the translation timestamps or literal values.
2. In a <field specification>, the table represented by the <translation table name> and the character pattern shown by the <field formatting specification>, determine the output format and translation for the given <field identifier>.
3. Valid <field formatting specification>s are as follows.

   Case:

   - `W`*num*—place the value in an output field of width *num*. The default field width is just large enough to contain the constant and a sign if specified. Truncation will occur on the right if the value is too large, and the field is left-justified. Truncation will occur on the left is the field is too large, and the field is right justified. Only one `W` specification is permitted for each <field formatting specification>.
   - `L`—place the value left-justified in the field. Cannot be specified with `R`.
   - `R`—place the value right-justified in the field. Right justification is the default. Cannot be specified with `L`.
   - `Z`—pad the field with zeros. Cannot be specified with `B`.
   - `B`—pad the field with blanks. Blank padding is the default. Cannot be specified with `Z`.

- `S`—include a sign character in the output. For negative numeric values the sign is always displayed. `S` forces a positive sign for positive numeric values. Cannot be specified for non-numeric data.

4. Within a <format string>, <less than operator> <less than operator> denotes a single <less than operator>.

5. Within a <format string>, <quote><quote> (that is, a <quote symbol>) denotes a single <quote>.

6. Any <character representation> appearing in the format string appears in an output string in the same relative position and order with respect to other <character representation>s and <field specification>s.

7. A <datetime string> is any sequence of characters not containing a single <quote>.

   Case:

   - The value represented by a <datetime string> is the special granule *beginning* if the <datetime string> is identical to the value of the *beginning_string* property.
   - The value represented by a <datetime string> is the special granule *forever* if the <datetime string> is identical to the value of the *forever_string* property.
   - The value of a <datetime string> is the special value *until changed* if the <datetime string> is identical to the value of the *until_changed_string* property.
   - The value represented by a <datetime string> is the special granule *initiation* if the <datetime string> is identical to the value of the *initiation_string* property. This datetime is the creation time of the schema for the database; no transaction time stored in this database can precedee this instant.
   - The value of a <datetime string> is the special value *now* if the <datetime string> is identical to the value of the *now_string* property. This special value, when bound in an executed statement, is identical to the value of `CURRENT_TIMESTAMP`.
   - The value represented by a <datetime string> is the value returned by a calendar if the <datetime string> is a contiguous subset of a string consitent with the value of the *determinate_datetime_format* property, which can include references to calendric-specific fields. The calendar named in the value of the *input_epoch_override* property is attempted first. If this calendar does not recognize one of the fields, the calendars are attempted in the order specified for the current calendric system.
   - Let $A$ be a valid <datetime string>, representing the datetime $B$. Let $T$ be a string consistent with the *time_zone_format* property, which can include references to the fields *minute* and *hour*. Let $TZ$ be an `INTERVAL HOUR TO MINUTE` computed from the values of the hour and minute fields. If the value of the *datetime_with_time_zone* property, with the *period* field replaced with $A$ and the *time_zone* field replaced with $B$, is identical to the <datetime string>, then the value represented by the <datetime string> is the datetime $B$ displaced by a time zone offset of $TZ$.
   - Let $A$ be a valid <datetime string>, representing the datetime $B$. Let $T$ be a string contained in the translation table named by the *time_zone_name_table*. Let $TI$ be the index associated with this string in this translation table. Let $TZ$ be an `INTERVAL HOUR TO MINUTE` computed by looking up $TI$ and $B$ in the system-wide timezone table provided by the DBA, with the schema `(INDEX SMALLINT, VALIDTIME PERIOD, ENDTIME TIMESTAMP, OFFSET INTERVAL HOUR TO MINUTE)`, where $B$ overlaps `VALIDTIME`. If the value of the *datetime_with_time_zone* property, with the *period* field replaced with $A$ and the *time_zone* field replaced with $B$, is identical to the <datetime string>, then the value represented by the <datetime string> is the datetime $B$ displaced by a time zone offset of $TZ$.

- Let $A$ and $B$ be valid <datetime string>s, representing the datetimes $C$ and $D$. Let $E$ be a string consistent with the *distribution_format* property, which can include references to the field *distribution_name*. If the value of the *indeterminate_datetime* property, with the *determinate_datetime_1* field replaced with $A$, the *determinate_datetime_2* field replaced with $B$, and the *distribution* field replaced with $E$, is identical to the <datetime string>, then the value represented by the <datetime string> is the indeterminate datetime with lower support $C$, upper support $D$, and distribution as named in $E$.
- Let $A$ be a valid <determinate interval string>, representing the interval $B$. Let $C$ be a string consistent with the *sign_format* property, which can include references to the field *sign*. If the value of the *now_relative_datetime_format* property, with the *now* field replaced with the value of the property *now_string*, the *determinate_interval* field replaced with $A$, and the *sign* field replaced with $C$, is identical to the <datetime string>, then the value represented by the <datetime string> is the now-relative datetime $now + B$ or $now$ - $B$, dependening on whether the *sign* field value is 0 or 1.
- Let $A$ be a valid <now-relative datetime string>, representing the datetime $B$. Let $C$ be a valid <determinate datetime string>, representing the datetime $D$. Let $E$ be a string consistent with the *distribution_format* property, which can include references to the field *distribution_name*. If the value of the *indeterminate_now_relative_datetime_format* property, with the *now_relative_datetime* field replaced with $B$, the *determinate_datetime* field replaced with $D$, and the *distribution* field replaced with $E$, is identical to the <indeterminate now-relative datetime string>, then the value represented by <indeterminate now-relative datetime string> is the indeterminate now-relative datetime with lower support $B$, upper support $D$, and distribution as named in $E$.
- Let $A$ be a valid <indeterminate interval string>, representing the interval $B$, with lower support $C$, upper support $D$, and distribution $E$. Let $F$ be a string consistent with the *sign_format* property, which can include references to the field *sign*. If the value of the *now_relative_with_indeterminate_interval_datetime_format* property, with the *now* field replaced with the value of the property *now_string*, the *indeterminate_interval* field replaced with $A$, and the *sign* field replaced with $F$, is identical to the <now-relative with indeterminate datetime string>, then the value represented by the <now-relative with indeterminate datetime string> is the indeterminate datetime with lower support $now + C$ or $now$ - $C$ depending on whether the *sign* field value is 0 or 1, upper support $D$, and distribution $E$.

8. An <interval string> is any sequence of characters not containing a single <quote>.

   Case:

   - The value of an <interval string> is the special value *all of time* if the <interval string> is identical to the value of the *all_of_time_string* property.
   - The value of an <interval string> is the special value *negative all of time* if the <interval string> is identical to the value of the *negative_all_of_time_string* property.
   - The value of an <interval string> is the value returned by a calendar if the <interval string> is a contiguous subset of a value consitent with the value of the *determinate_interval_format* property, which can include references to calendric-specific fields. The calendar named in the value of the *input_epoch_override* property is attempted first. If this calendar does not recognize one of the fields, the calendars are attempted in the order specified for the current calendric system.
   - Let $A$ and $B$ be valid <determinate interval string>s, representing the intervals $C$ and $D$. Let $E$ be a string consistent with the *distribution_format* property, which can include references to the field *distribution_name*. If the value of the *indeterminate_interval* property, with the *determinate_interval_1* field replaced with $A$, the *determinate_interval_2* field replaced with $B$, and the *distribution* field replaced with $E$, is identical to the <interval string>, then the

value represented by the <interval string> is the indeterminate interval with lower support $C$, upper support $D$, and distribution as named in $E$.

- Let $A$ be a valid <determinate datetime string>, representing the datetime $B$. Let $C$ be a string consistent with the *sign_format* property, which can include references to the field *sign*, whose value is restricted to being 1. If the value of the *now_relative_interval_format* property, with the *now* field replaced with the value of the property *now_string*, the *datetime* field replaced with $A$, and the *sign* field replaced with $C$, is identical to the <now-relative interval string>, then the value represented by the <now relative interval string> is the now-relative interval *now* - $B$.

9. A <period string> is any sequence of characters not containing a single <quote>.

   Case:

   - The value of a <period string> is the special value *all of time* if the <period string> is identical to the value of the *all_of_time_period_string* property.
   - Let $A$ and $B$ be valid <datetime string>s, representing datetimes $C$ and $D$. If the value of the *determinate_period_format* property, with the *determinate_datetime_1* field replaced by $A$ and the *determinate_datetime_2* field replaced by $B$, is identical to the <period string>, then the value of the <period string> is the period from $C$ to $D$.
   - Let $A$ be a valid <period string>, representing the period $B$. Let $T$ be a string consistent with the *time_zone_format* property, which can include references to the fields *minute* and *hour*. Let $TZ$ be an `INTERVAL HOUR TO MINUTE` computed from the values of the hour and minute fields. If the value of the *period_with_time_zone* property, with the *period* field replace with $A$ and the *time_zone* field replaced with $B$, is identical to the <datetime string>, then the value represented by the <period string> is the period $B$ displaced by a time zone offset of $TZ$.
   - Let $A$ be a valid <datetime string>, representing the datetime $B$. Let $T$ be a string contained in the translation table named by the *time_zone_name_table*. Let $TI$ be the index associated with this string in this translation table. Let $TZ$ be an `INTERVAL HOUR TO MINUTE` computed by looking up $TI$ and $B$ in the system-wide timezone table provided by the DBA, with the schema `INDEX SMALLINT, VALIDTIME PERIOD, ENDTIME TIMESTAMP, OFFSET INTERVAL HOUR TO MINUTE)`, where $B$ overlaps `VALIDTIME`. If the value of the *period_with_time_zone* property, with the *period* field replace with $A$ and the *time_zone* field replaced with $B$, is identical to the <datetime string>, then the value represented by the <period string> is the period $B$ displaced by a time zone offset of $TZ$.

10. The data type of a <period literal> is `PERIOD`.

11. The *starting_delimiter* and *ending_delimiter* fields mentioned within the *determinate_period_format* determine whether the period literal is closed-closed, closed-open, open-closed, or open-open. In any case, the value is stored internally as a closed-closed period.

12. The non-terminal <calendric-property specification> is defined in Appendix 16.1.

13. If <calendric-property specification> contains a <calendric-spec clause> then the calendric system named in the <calendric-spec clause> is used when interpreting this literal. Otherwise, the globally declared calendric system whose scope includes this literal is used.

14. If <calendric-property specification> contains a <property-spec clause> then the properties contained in the named property table are activated before interpreting this literal, and deactivated after interpreting this literal.

15. If no `DECLARE CALENDRIC SYSTEM` command has been entered then the implementation defined default calendric system is assumed.

Additional general rules:

1. The precision of a <time literal> is `SECOND(0)` if <time precision> is not present. Otherwise, it is that specified by <time precision>.
2. The precision of a <timestamp literal> is `SECOND(0)` if <time precision> is not present. Otherwise, it is that specified by <timestamp precision>.
3. The granule denoted by a <datetime literal> is assumed to be the first granule represented by the datetime string. This behavior may be changed with appropriate field names.
4. 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. This behavior may be changed with appropriate field names.
5. Closed-closed periods are closed on both ends (i.e., the period includes both specified datetimes). Closed-open periods do not contain their specified ending datetime; they terminate one granule before their ending datetime. Similarly, open-closed periods do not contain their specified starting datetime, and open-open do not contain either their specified starting or ending datetimes.
6. If the current calendric system is unable to successfully interpret a datetime, period, or interval literal then an exception condition is raised: *data exception—invalid time value literal.*

### 10.3 Section 5.4 Names and identifiers

The following productions are added.

<calendric system name> ::=
- <identifier>

<property table name> ::=
- <table name>

<time granularity> ::=
- <identifier>

Additional syntax rules:

1. The identifiers denoting calendric systems and property tables are implementation dependent.
2. The available <time granularity>s are implementation dependent, but must include `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, and `SECOND`.

# 11 Section 6 Scalar Expressions

## 11.1 Section 6.1 <data type>

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

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

<period type> ::=
- [ <indeterminate data type> ] `PERIOD` [ <period precision> ] [ `WITH TIME ZONE` ]

<period precision> ::=
- <period qualifier>

The production, <indeterminate data type> is added.

<indeterminate data type> ::=
- [ `NONSTANDARD` ] [ `GENERAL` ] `INDETERMINATE`

<surrogate type> ::=
- `SURROGATE`

The <datetime type> non-terminal is modified.

<datetime type> ::=
[ <indeterminate data type> ] `DATE`
- | [ <indeterminate data type> ] `TIME` [ <time precision> ] [ `WITH TIME ZONE` ]
- | [ <indeterminate data type> ] `TIMESTAMP` [ <timestamp precision> ] [ `WITH TIME ZONE` ]

<time precision> ::=
- <left paren> <time fractional seconds precision> <right paren>

<timestamp precision> ::=
- <timestamp qualifier>

<interval type> ::=
- [ <indeterminate data type> ] `INTERVAL` [ <interval qualifier> ]

Additional general rules:

1. The delimiting datetimes of a period shall have the same precision and scale.
2. Values of type `SURROGATE` cannot be seen (displayed). Consequently, attributes of `SURROGATE` type are not allowed in the outermost `SELECT` clause of a query. Also, attributes of surrogate type cannot be assigned an explicit value.
3. A special reserved word, `NEW` may be used when updating an attribute value of `SURROGATE` type. The new value is a previously unused value.

4. Values of type `SURROGATE` can only be compared with respect to identity.
5. The default distribution is standard (not `NONSTANDARD`).
6. The default indeterminate datetime is compact (not `GENERAL`).
7. The default datetime is determinate (not `INDETERMINATE`).
8. The size of the timestamp format allocated depends on the kind of timestamp selected and the user-specified precision. Enough space must be allocated to the data fields to accommodate the precision of the timestamp (precision rules are described elsewhere). The default indeterminate timestamp format is the chunked with standard distributions format. By specifying `GENERAL` the user chooses to use one of the general, indeterminate timestamp formats. By specifying `NONSTANDARD` the user chooses to use one of the nonstandard timestamp formats.

### 11.2 Section 6.2 <value specification> and <target specification>

The productions for the non-terminals <parameter specification> and <variable specification> are augmented to allow calendric system and property selection per-item.

<parameter specification> ::=
- <parameter name> [ <indicator parameter> ] [ <calendric-property specification> ]

<variable specification> ::=
- <embedded variable name> [ <indicator variable> ] [ <calendric-property specification> ]

Additional syntax rules:

1. The non-terminal <calendric-property specification> is defined in Appendix 16.1.
2. If <calendric-property specification> is specified then <parameter name> must have the data type <character string type>. Similar remarks apply to <embedded variable name>.
3. If <calendric-property specification> is specified then the value contained in <parameter name> or <variable name> is interpreted as a temporal value according to the calendric system and/or calendar properties named by the <calendric-property specification>.
4. If <calendric-property specification> contains a <calendric-spec clause> and the data type of the column corresponding to the <parameter specification> or <variable specification> is `DATE`, `TIME`, `TIMESTAMP`, `PERIOD`, or `INTERVAL`, then the calendric system named in the <calendric-spec clause> is used to translate the timestamp into a temporal value.
5. If <calendric-property specification> contains a <property-spec clause> and the data type of the column corresponding to the <parameter specification> or <variable specification> is `DATE`, `TIME`, `TIMESTAMP`, `PERIOD`, or `INTERVAL`, then the property table named in the <property-spec clause> are activated before translating the timestamp, and deactivated immediately after translating the timestamp.
6. If no `SET CALENDRIC SYSTEM` command has been entered then the implementation defined default calendric system is assumed.

### 11.3 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> [ `WITH CREDIBILITY` <integer> ]
  | <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 Element is assumed.
6. If <partitioning unit> is not specified, then no partitioning is assumed.
7. Partitioning on `INSTANT` is only allowed for event 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>.
4. The credibility is a value between 0 and 100 (inclusive).
5. If the credibility phrase is missing, the default credibility is 100 or as specified by the user with a set statement.

### 11.4 Section 6.5 <set function specification>

An optional clause to the general set function production was added for weighted aggregates.

<general function type> ::=
<set function type> <left paren> [ <set quantifier> ]
• [ `WEIGHTED` ]
<value expression> <right paren>

One aggregate was added to the set function type.

<set function type> ::=
• | `RISING`

Additional syntax rules:

1. Let $DT$ be the data type of the <value expression>.
2. If `RISING` is specified, the data type of the result is a period.
3. If `SUM` is specified, DT shall not be an instant or a period.
4. If `AVG` is specified, DT shall not be a period, an event set, or a temporal element.
5. If `COUNT` is specified, `WEIGHTED` has no effect.

Additional general rules:

1. If `WEIGHTED` is specified, and DT is temporal, then `WEIGHTED` has no effect on the aggregate.
2. If `WEIGHTED` is specified, let $A$ be the specified attribute of the aggregate and let $T$ be the argument source.

   Case:

   (a) If `MAX` is specified, then the result is attribute A of the tuple, where, of T, attribute A multiplied by the number of granules in its timestamp is maximal.
   (b) If `MIN` is specified, then the result is attribute A of the tuple, where, of T, attribute A multiplied by the number of granules in its timestamp is minimal.
   (c) If `SUM` is specified, then the result is the sum of all attributes A in T, piecewise multipled by their timestamps, divided by the sum of the timestamps.
   (d) If `AVG` is specified, then the result is the SUM function over T divided by the cardinality of T.

3. If `RISING` is specified without `WEIGHTED`, then the result shall be the largest period such that the argument source T is monotonic increasing. If `WEIGHTED` is specified, then the largest period is computed over the value of each attribute multipled by its timestamp.
4. 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.

5. 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.*

### 11.5 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>
- | `SCALE` <left paren> <datetime value expression> `AS` <time granularity> <right paren>
- | `NOBIND` <left paren> <datetime literal> <right paren>
- | `NOBIND` <left paren> <column reference> <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 or bitemporal event tables which are partitioned.
3. Local invocation of a scale function overrides the global default.
4. The granularity of the resulting type of the `SCALE` operation is <time granularity>.
5. A `NOBIND` function can only appear in the target list of an `insert` or `modify` statement. Any other use of a nobind will generate a compile-time error.

### 11.6 Section 6.10 <cast specification>

Casting to different granularities is allowed, by adding to the options of the <cast target>.

<cast target> ::=
<domain name>
| <data type>
• | <time granularity>

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), instant set (IS) data types, and to add the time granularity (G) cast target.

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

2. If *SD* is AN and *TD* is YM or DT then the conversion is first done to the EN type.

3. If *SD* is EN and *TD* is YM or DT then the conversion is dependent on the current calendric system in effect when the <cast specification> is executed.

4. If *SD* is C and *TD* is D, T, TS, YM, DT, or P then the conversion is dependent on the current calendric system and set of input properties in effect when the <cast specification> is executed.

   Let *CS* be the current calendric system and *PS* be the appropriate output format string currently in effect when the <cast specification> is executed. Then the <cast specification> `CAST(C AS` *X*`)` where *X* is D, T, TS, YM, DT, or P is equivalent to the following.

   *C* `WITH CALENDRIC` *CS* `WITH PROPERTIES` *PS*

5. If *SD* is D, T, TS, YM, DT, or P and *TD* is FC or VC then the conversion is dependent on the current calendric system and set of output properties in effect when the <cast specification> is executed, as described in Section 5.3 <literal>.

6. If *SD* is D, T, or TS and *TD* is P then the conversion results in a period of duration one granule.

7. If *SD* is C and *TD* is P then the conversion is first done to the T data type.

8. Let *CS* be the current calendric system and *PS* be the appropriate output format string currently in effect when the <cast specification> is executed. Then the <cast specification> `CAST(`*T* `AS` *X*`)` where *T* is D, T, TS, TM, DT, or P and *X* is VC or FC is equivalent to the following.

*T* `WITH CALENDRIC` *CS* `WITH PROPERTIES` *PS*

9. If *SD* is YM or DT and *TD* is EN or AN then the conversion is dependent on the current calendric system in effect when the <cast specification> is executed.
10. If *SD* is C, D, T, or TS and *TD* is TE then the conversion is first done to the *P* type.
11. If *SD* is P and *TD* is TE, then the conversion is into a temporal element containing one period.
12. If *SD* is C, T, TS, or P and *TD* is IS then the conversion is first done to the *TE* type.
13. If *SD* is TE and *TD* is IS then the conversion is done by applying `FIRST` to each period in the set.
14. If *SD* is D, T, TS, YM, DT, P, TE, or IS and *TD* is G then the conversion results in a value of the data type *SD* at the underlying granularity *TD*.

Additional general rules:

1. Rule 3(c) is replaced with the following.

   If *TD* is exact numeric and *SD* is interval then if there is a representation of *SV* in the type *TD* that does not lose any leading significant digits then *TV* is that representation. Otherwise, an exception condition is raised: *data exception—numeric value out of range.*

2. Rule 5(e) is replaced with the following.

   If *SD* is a datetime, interval or period then let *Y* be the calendar dependent character string produced from *SV* such that the interpreted value of *Y* is *SV* and the interpreted precision of *Y* is the precision of *SD*.

   Case:

   - If *Y* contains any <SQL language character> that is not in the repertoire of *TD* then an exception condition is raised: *data exception—invalid character value for cast.*
   - If the length in characters *LY* of *Y* is equal to *LTD* then *TV* is *Y*.
   - If the length in characters *LY* of *Y* is less than *LTD* then *TV* is extended on the right by $LTD - Y$ spaces.
   - Otherwise an exception condition is raised: *data exception—string data, right truncation.*

3. Rule 6(e) is replaced with the following.

   If *SD* is a datetime, interval or period then let *Y* be the calendar dependent character string produced from *SV* such that the interpreted value of *Y* is *SV* and the interpreted precision of *Y* is the precision of *SD*.

   Case:

   - If *Y* contains any <SQL language character> that is not in the repertoire of *TD* then an exception condition is raised: *data exception—invalid character value for cast.*
   - If the length in characters *LY* of *Y* is less than or equal to *MLTD* then *TV* is *Y*.
   - Otherwise an exception condition is raised: *data exception—string data, right truncation.*

4. Rules 9(b)–(c) are deleted, and Rule 9(a) is replaced with the following.

   If *TD* is a datetime data type and *SD* is a character string then the determination of *TV* from *SV* is calendar dependent. If *TV* cannot be determined from *SV* then an exception condition is raised: *data exception—invalid character value for cast.*

5. Rules 10 and 11 are deleted.

6. Rule 12 is replaced with the following.

   If *TD* is an interval data type, then

   Case:

   (a) If *SD* is exact numeric then the determination of *TV* from *SV* is calendar dependent.

      If the representation of *SV* in the data type *TD* would result in the loss of leading significant digits, then an exception condition is raised: *data exception—time value out of range.*

   (b) If *SD* is character string then the determination of *TV* from *SV* is calendar dependent.

      If *TV* cannot be determined from *SV* then an exception condition is raised: *data exception—invalid character value for cast.*

   (c) If *SD* is P and *TD* is YM or DT, then the duration of *SD* is determined in the precision of *TD*.

### 11.7 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>

## 11.8 Section 6.14 <datetime value expression>

The <time zone specifier> is augmented to allow symbolic time zones, such as `'MST'`. The production for the non-terminal <period primary> is augmented to also include references to tables themselves. Also, expressions evaluating to temporal expressions are added.

<time zone specifier> ::=
    `LOCAL`
  | `TIME ZONE` <interval value expression>
• | `TIME ZONE` <character string literal>

| *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: Valid Arithmetic Expressions and Results.

Additional general rules:

1. If <character string literal> is a string contained in the translation table named by the *time_zone_-name*. Let $TI$ be the index associated with this string in this translation table. Let $B$ be the value of <datetime primary>. Let $TZ$ be an `INTERVAL HOUR TO MINUTE` computed by looking up $TI$ and $B$ in the system-wide timezone table provided by the DBA, with the schema `INDEX SMALLINT, VALIDTIME PERIOD, ENDTIME TIMESTAMP, OFFSET INTERVAL HOUR TO MINUTE`), where $B$ overlaps `VALIDTIME`.

2. The following is added to Rule 3.

   The semantics of <datetime value expression>s involving <period term>s is calendar-dependent. If the underlying granularities of both are supplied by the `SQL92` calendar, then the semantics are as follows. (Original Rule 3 goes here.)

3. Operands are coerced to the global scale/cast specified in the last `SET SCALE/SET CAST` command prior to the operation. If no such command was issued or the defaults are specified, then operands are scaled as needed to enforce ledt-operand semantics.

4. The range of intermediate results is the maximum allowed by the implementation.

5. The following is added to Rule 6.

   If <datetime value expression> is specified, the semantics is calendar-dependent. If the underlying granularities of both the <datetime value expression> and the <datatime term>, as well as the <period qualifier> are supplied by the `SQL92` calendar, then the semantics are as follows. (Original Rule 6 goes here.)

## 11.9 Section 6.15 <interval value expression>

The following production is added to the <interval value expression> non-terminal.

<interval value expression> ::=
- | <interval value function>

New general rules:

1. The following is added to Rule 6.

   If <datetime value expression> is specified, the semantics is calendar-dependent. If the underlying granularities of both the <datetime value expression> and the <datatime term>, as well as the <period qualifier> are supplied by the `SQL92` calendar, then the semantics are as follows. (Original Rule 6 goes here.)

### 11.10 Section 6.?? <interval value function>

This is a new section.

<interval value function> ::=

- `INTERVAL` <left paren> <period value expression> <right paren>
- | `ABSOLUTE` <left paren> <interval value expression> <right paren>
- | `SCALE` <left paren> <interval value expression> `AS` <time granularity> <right paren>
- | `NOBIND` <left paren> <interval literal> <right paren>
- | `NOBIND` <left paren> <column reference> <right paren>

Additional general rules:

1. Local invocation of a scale function overrides the global default.
2. The granularity of the resulting type of the `SCALE` operation is <time granularity>.
3. A `NOBIND` function can only appear in the target list of an **`insert`** or **`modify`** statement. Any other use of a nobind will generate a compile-time error.

### 11.11 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>

<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 valid.

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.*

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

This is a new section.

<period value function> ::=

- `VALID` <left paren> { <table name> | <correlation name> } <right paren>
- | `TRANSACTION` <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>
- | `SCALE` <left paren> <period value expression>
- | `NOBIND` <left paren> <period literal> <right paren>
- | `NOBIND` <left paren> <column reference> <right paren>

Additional general rules:

1. Use of `VALID` is allowed only on valid time state or bitemporal state tables that are partitioned, and denotes a maximal period in the timestamp of the underlying tuple.
2. Use of `TRANSACTION` is allowed only on transaction or bitemporal tables, and denotes a maximal period in transaction time when the values of the columns and the valid time associated with the tuple remained constant.
3. `FIRST` (`LAST`) extracts the first (last) maximal period from the temporal element.
4. Local invocation of a scale function overrides the global default.
5. The granularity of the resulting type of the `SCALE` operation is <time granularity>.
6. A `NOBIND` function can only appear in the target list of an `insert` or `modify` statement. Any other use of a nobind will generate a compile-time error.

### 11.13 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).

### 11.14 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>
- | `SCALE` <left paren> <temporal element expression> `AS` <time granularity> <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 or bitemporal state table that has not been partitioned.
2. Intersection of temporal elements is set intersection.
3. Local invocation of a scale function overrides the global default.
4. The granularity of the resulting type of the `SCALE` operation is <time granularity>.

## 11.15 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).

### 11.16 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 or bitemporal event table that has not been partitioned.

## 12 Section 7 Query expressions

### 12.1 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> }

## 12.2 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 granularity.

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

2. The default `VALID` clause is

   `VALID INTERSECT PERIOD 'all of time'`.

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.

### 12.3 Section 7.6 <where clause>

To the production for <where clause> is added the plausibility phrase.

<where clause> ::=
- `WHERE` <search condition> [ `WITH PLAUSIBILITY` <integer> ]

Additional general rules:

1. The plausibility is a value between 1 and 100 (inclusive). A value of 1 implies a non-zero plausibility less than 1.
2. If the plausibility phrase is missing, the default plausibility is 100 or as specified by the user with a set statement.

## 12.4 Section 7.7 <group by clause>

The production for grouping column reference is extended.

```
<grouping column reference> ::=
            <column reference> [ <collate clause> ]
•         | <temporal partition>

<temporal partition> ::=
•         { VALID <left paren> { <table name> | <correlation name> } <right paren>
•           | <column reference> }
•           [ USING { <partition expression> | INSTANT } ]
•           [ LEADING <partition expression> ]
•           [ TRAILING <partition expression> ]

<partition expression> ::=
•           <integer>
•         | <time granularity granularity>
•         | <integer> <time granularity>
•         | PERIOD 'All of time' <time granularity>
```

Additional syntax rules:

1. If the using clause, or the leading clause or the trailing clause is present, and `VALID` is not present, then the type of the <column reference> in a <temporal partition> clause must be a timestamp. If the <column reference> is simply a timestamp with no leading, trailing, of using clause, then partition the relation as SQL-92 defines.
2. `VALID` associated with a particular table may only be present once in a <group by clause>

Additional general rules:

1. If the special period `PERIOD 'All of time'` <time granularity> is present in the using clause, then the partition includes all of the time-line. If the leading clause (trailing) includes the `PERIOD 'All of time'` <time granularity>, then the leading partition (trailing) is of maximal length (i.e. includes all previous (later) granules on the time-line).
2. The granularity of the using, leading, and trailing clauses, if they are present, must be the same as the granularity of the table.
3. If the type of the <column reference> in a <temporal partition> clause is a timestamp, or `VALID` is present, then

   Case:

   (a) If the using clause is not present, then the default is `INSTANT` for clauses which contain `VALID`, and `PERIOD 'All of time'` <time granularity> for <column references> in a <temporal partition>. The default granularity is the table's granularity. The using and leading clauses may only specify integral multiples of this granularity.

   (b) If the leading (or trailing) clause is not present, then the default length of the missing clause is length 0.

   (c) The granularity in the leading, trailing, and using clauses is a calendar granularity.

   (d) If a granularity is given without the accompanying integer length, the length is assumed to be 1.

4. If any or all of the using, trailing or leading clauses are present, or `VALID` is present, then partition the table the following way. These computations are done at the underlying granularity of the valid clause. The result of the <temporal partition> will be an assignment of tuples to one or more granules in the query result's valid time-line. Then an aggregate value will be computed over the set of tuples associated with each granule. For each tuple, we determine which granules it associated with in the following way. Also, if the <column reference> is a timestamp, then in the following, use the values of the timestamp instead of the valid time from that relation when processing the <temporal partition> which contains that <column reference>.

   (a) For each <temporal partition> (with $R$ the expression's <table name> or <correlation name>), determine which granules the tuple overlaps. This is done by computing the least ($L$) and greatest granules ($G$) which overlap the tuple's valid time from $R$, in the granularity of the valid clause, with respect to the leading, trailing, and using clauses. The tuple is first *tentativly associated* with the sequence of granules from $L$ to $G$, inclusive.

   (b) The using clause specifies how many consecutive granules ( $\{ g_1, \ldots, g_{\mathrm{n}} \}$ ) are to be considered for each partition. The leading and trailing clauses extend this sequence by their integral amounts, respectively to the beginning and the end of the sequence. A tuple overlaps all granules in a partition if it's valid time with respect to $R$ intersects $\{ g_1, \ldots, g_{\mathrm{n}} \}$.

   (c) If for all <temporal parition>'s, a tuple is tentatively associated to a granule $g$, then the tuple is associated with $g$.

## 12.5 Section 7.8 <having clause>

Additional general rules:

1. Let $T$ be one of the clauses in the <temporal partition> clause.
2. If $T$ contains a using clause, then the using clause must be larger than a single granule, and the leading and trailing clauses must be zero length.
3. If the group-by clause contains a <temporal partition>, then the result of a reference to valid time in the having clause is the valid time of the group defined by the <temporal partition>.

## 12.6 Section 7.9 <query specification>

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

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

We add an option to indicate use of the completed schema.

<select list> ::=<br>
<column list><br>
| <asterisk><br>
• | <asterisk> <asterisk><br>
| <select sublist> [ { <comma> <select sublist> }... ]

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>.
2. Specification of the ** option results in the use of the completed schemes for the table(s) specified.

# 13 Section 8 Predicates

## 13.1 Section 8.1 <predicate>

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

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

## 13.2 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.

### 13.3 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.

## 13.4 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

# 14 Section 10 Additional Common Elements

## 14.1 Section 10.1 <interval qualifier>

This is significantly generalized to allow implementation-defined granularities. The <non-second date-time field> non-terminal is removed, <timestamp qualifier> and <period qualifier> are added, and the following non-terminals are modified.

<start field> ::=

- <time granularity> [ <left paren> <interval leading field precision> <right paren> ]
- | <left paren> <interval string> <interval qualifier> <right paren>

<end field> ::=

- <time granularity> [ <left paren> <interval fractional seconds precision> <right paren> ]

<single datetime field> ::=

- <time granularity> [ <left paren> <interval leading fixed position>
  [ <comma> <interval trailing field position> ] <right paren> ]

<timestamp qualifier> ::=

- [ <start field> `TO` ] <end field>
- | <single datetime field>

<period qualifier> ::=

- <timestamp qualifier>

The general rules are significantly generalized to remove fairly arbitrary restrictions.

# 15 Section 11 Schema definition and manipulation

We add to the production for <schema element> to allow dynamic definition of distributions.

<schema element> ::=
- | <create distribution statement>

### 15.1 Section 11.3 <table definition>

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

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

Three productions are added.

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

<vacuuming definition> ::=
• `VACUUM` <datetime value expression>

Additional general rules:

1. Case:
   (a) if neither `VALID` nor transaction is specified, the table is a snapshot table.
   (b) If `AS VALID STATE` is specified, and `TRANSACTION` is not 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, and `TRANSACTION` is not specified, then the tuples are timestamped with valid-time instant sets. The precision and scale of the instants can be specified.
   (d) If `TRANSACTION` is specified, and `VALID` is not specified, then the tuples are timepstamped with transaction-time elements. The scale of the timestamps is implementation-dependent.
   (e) If `TRANSACTION` and `VALID STATE` are specified, the the tuples are timestamped with bitemporal elements that are sets of bitemporal chronons. The precision and scale of the valid-time dimension can be specified; the scale of the transaction-time dimension is implementation-dependent.
   (f) If `TRANSACTION` and `VALID EVENT` are specified, the the tuples are timestamped with bitemporal instant sets that are sets of bitemporal chronons. The precision and scale of the valid-time dimension can be specified; the scale of the transaction-time dimension is implementation-dependent.
2. The <vacuuming definition> is only allowed when the table supports transaction time.
3. If <vacuuming definition> is not specified, `VACUUM TIMESTAMP CURRENT_TIMESTAMP` is assumed (the default).

## 15.2 Section 11.4 <column definition>

```
<column definition> ::=
        <column name> {<data type>
•                [ INAPPLICABLE <value expression> ]
                 |<domain name> }
        [ <default clause> ]
        [ <column constraint definition>... ]
        [ <collate clause> ]
```

Additional General Rules:

1. The INAPPLICABLE clause expressions may be either simple of a function only of the attributes in the completed schema for the table.

### 15.3 Section 11.5 <default clause>

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

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

Additional syntax rules:

1. If <datetime value function>, <period value function>, or <interval value function> is specified then any parameters passed to these functions must be property values representing a special time value or literal values.

2. Let $T$ be the type of the column being initialized.

3. If $T$ is `DATE`, `TIME`, `TIMESTAMP`, `PERIOD`, or `INTERVAL` then `USER` and `SYSTEM USER` may not be specified.

4. If $T$ is `DATE`, `TIME` or `TIMESTAMP` then either a <literal> representing a <datetime literal> or a <datetime value function> may be specified. The calendric system used to interpret the constant is the calendric system whose scope is the smallest scope which encompasses the literal. The properties used to interpret the constant are the set of properties active when the default clause is executed.

5. If $T$ is `PERIOD` then either a <literal> representing a <period literal> or a <period value function> may be specified. The calendric system used to interpret the constant is the calendric system whose scope is the smallest scope which encompasses the literal. The properties used to interpret the constant are the set of properties active when the default clause is executed.

6. If $T$ is `INTERVAL` then either a <literal> representing an <interval literal> or an <interval value function> may be specified. The calendric system used to interpret the constant is the calendric system whose scope is the smallest scope which encompasses the literal. The properties used to interpret the constant are the set of properties active when the default clause is executed.

### 15.4 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 def>
- | <add transaction definition>
- | <delete transaction definition>
- | <scale valid definition>
- | <cast valid definition>
- | <alter vacuuming 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> ]

<add transaction definition> ::=
- `ADD TRANSACTION`

<drop transaction definition> ::=
- `DROP TRANSACTION`

<scale valid definition> ::=
- `SCALE VALID AS` <timestamp precision>

<cast valid definition> ::=
- `CAST VALID AS` <timestamp precision>

<alter vacuuming definition> ::=
- `VACUUM` <datetime value expression>

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 or bitemporal table.
4. For the <replace valid definition>, T shall be a valid-time or bitemporal table.
5. For the <add transaction definition>, T shall be a snapshot table or a valid-time table.
6. For the <drop transaction definition>, T shall be a transaction-time or bitemporal table.
7. For the <scale valid definition>, T shall be a valid-time or bitemporal table.

8. For the <cast valid definition>, T shall be a valid-time or bitemporal table.
9. For the <alter vacuuming definition>, T shall be a transaction-time or bitemporal 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. In the <replace valid definition>, scale or cast is used as specified by the <set scale statement> or <set cast statement>.
3. For <scale valid definition>, the temporal element of each tuple of T is converted to the new precision and scale, using a cast or scale operation.
4. 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.
5. 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)
   ```

6. 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.
7. 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)
   ```

8. The <datetime value expression> must, when the <alter table statement> is issued, evaluate to a time value that is either not before the current cut-off point or is after the current time.
9. When an <alter table statement> with an <add transaction time> clause, but with no <alter vacuuming definition>, is applied to a table that does not support transaction time, the time the <alter table statement> takes effect is used as the cut-off point of the altered table.

## 15.5 Section 11.?? Distributions

This is a new section.

<create distribution statement> ::=
- `CREATE [ { GLOBAL | LOCAL } TEMPORARY ] DISTRIBUTION`
  <distribution name> `USING` <table name>

<alter distribution statement> ::=
- `ALTER DISTRIBUTION` <distribution name> `USING` <table name>

<drop distribution statement> ::=
- `DROP DISTRIBUTION` <distribution name>

Additional general rules:

1. The distribution must conform to implementation-dependent distribution constraints, otherwise an exception is raised.
2. The <create distribution statement> establishes a new distribution name.
3. Altering a distribution effectively destroys the old distribution and replaces it with a new distribution having the indicated table descriptor.

## 16 Section 12 Module

The production for the non-terminal <module contents> is changed to include a global calendric system declaration statement, and a new non-terminal <declare calendric system> is added to define this statement.

<module contents> ::=
<declare cursor>
• | <declare calendric system>
| <dynamic declare cursor>
| <procedure>

<declare calendric system> ::=
• `DECLARE CALENDRIC SYSTEM WITH` <calendric spec>

<calendric spec> ::=
• `DEFAULT`
• | <calendric system name>

Additional syntax rules:

1. In a sequence of SQL statements, the last calendric system specified in a `DECLARE CALENDRIC SYSTEM` command remains in effect until a new `DECLARE CALENDRIC SYSTEM` command is entered.
2. A `DECLARE CALENDRIC SYSTEM WITH CALENDRIC DEFAULT` statement reactivates the implementation defined default calendric system.

### 16.1 Section 12.5 <SQL procedure statement>

The production for the non-terminal <SQL session statement> is changed to include a session-level calendric system selection command, default session-level scale and align specification commands, and an additional production is added to define the calendric system selection command. We also add an option to indicate which schema version to use.

<SQL session statement> ::=
    <SQL set identifier statement>
  | <set constraints mode statement>
  | <set transaction statement>
• | <set properties statement>
• | <set scale statement>
• | <set cast statement>
• | <set credibility statement>
• | <set plausibility statement>
• | <create distribution statement>
• | <alter distribution statement>
• | <drop distribution statement>
• | <schema set statement>

<set properties statement> ::=
• `SET PROPERTIES`
• [ `FOR CHARACTER SET` [ `DEFAULT` | `NATIONAL` | <character set> ] ]
• [ `FOR` { <time granularity> | <calendar name> }]
  `WITH` <property spec>

<calendric-property specification> ::=
• [ <calendric-spec clause> ] [ <property-spec clause>... ]

<calendric-spec clause> ::=
• `WITH CALENDRIC` <calendric spec>

<property-spec clause> ::=
• `WITH PROPERTIES` <property spec>

<property spec> ::=
• `PREVIOUS`
• | `DEFAULT`
• | <property table name>
• | <table value constructor>

<schema set statement> ::=
• `SET SCHEMA` <datetime value expression>

<set credibility statement> ::=
• `SET CREDIBILITY` { <integer> | `AS DEFAULT` }

<set plausibility statement> ::=
• `SET PLAUSIBILITY` { <integer> | `AS DEFAULT` }

<set scale statement> ::=
• `SET SCALE` {<time granularity> | `AS DEFAULT` }

<set cast statement> ::=

- SET CAST $\{$<time granularity> $|$ AS DEFAULT $\}$

Additional syntax rules:

1. The non-terminal <calendric system name> must be an <identifier> naming a calendric system.
2. The non-terminal <property table name> is the name of a property table defining properties for the specified character set.
3. The non-terminal <table value expression> enumerates the rows of a property table.
4. The most recent invocation of a <set credibility statement> or a <set plausibility statement> takes precedence.
5. If both the <set credibility statement> and the <set plausibility statement> are omitted, then the defaults, 100 and 100, respectively, are assumed.
6. The most recent invocation of a <set scale statement> or a <set cast statement> takes precedence.
7. If both the <set scale statement> and the <set cast statement> are omitted (or specified *as default*, then left argument granularity semantics is assumed.
8. Case:
   - If neither <time granularity> nor <calendar name> is specified, then the properties for all granularities are altered.
   - If <time granularity> is specified, then only the properties for that granularity are altered.
   - If <calendar name> is specified, then only the properties for the granularities defined by that calendar are altered.
9. The <datetime value expression> evaluates to a transaction-time instant that identifies a particular schema version.

Additional general rules:

1. Specifying `SET PROPERTIES WITH PREVIOUS` causes the previous set of active properties for the specified character set to be reactivated.
2. Specifying `SET PROPERTIES WITH DEFAULT` causes the implementation defined set of default properties for the specified character set to be activated.
3. A property table must have the schema (property:*character string*, value:*character string*). The command to create a persistent property table with property values of length at most twenty characters is the following.

   ```
   CREATE TABLE property_table(property VARCHAR 20, value VARCHAR 20)
   ```

4. If a <set properties statement> or <property-spec clause> names a non-existent <property table name>, then an exception condition is raised: *data exception—property table non-existent.*

# 17 Section 13 Data manipulation

## 17.1 Section 13.3 <fetch statement>

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

### 17.2 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>.

### 17.3 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.

## 17.4 Section 13.8 <insert statement>

The <insert column list> is modified to permit the use of the `NEW` reserved word.

<insert column list> ::=
- <insert column> [ { <comma> <insert column> }... ]

The <insert column> is a new production.

<insert column> ::=
- <column name>
- | `NEW`

Additional general rules:

1. `NEW` is permitted only when the <data type> of the corresponding column is `SURROGATE`.

### 17.5 Section 13.9 <update statement: positioned>

Additional general rules:

1. If $T$ is a transaction-time or bitemporal table, the transaction time of the appended or update tuple is `PERIOD(TIMESTAMP CURRENT_TIMESTAMP, NOBIND(TIMESTAMP 'until changed'))`.

### 17.6 Section 13.10 <update statement: searched>

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

Additional general rules:

1. If $T$ is a transaction-time or bitemporal table, the transaction time of the appended or update tuple is `PERIOD(TIMESTAMP CURRENT_TIMESTAMP, NOBIND(TIMESTAMP 'until changed'))`.

# 18 Section 21 Information Schema and Definition Schema

## 18.1 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'))
ALTER TABLE TABLES ADD COLUMN
       TRANSACTION_TIME CHARACTER_DATA
             CONSTRAINT TRANSACTION_TIME_CHECK
                        CHECK (TRANSACTION_TIME IN ('STATE', 'NONE'))
ALTER TABLE TABLES ADD COLUMN
       VACUUM_CUT-OFF   TIMESTAMP
```

The precision and scale of the VACUUM_CUT_OFF column is implementation-defined.

## 18.2 Section 21.3.?? TEMPORAL_SPEC base table

```
CREATE TABLE                 TEMPORAL_SPEC {
       TABLE_NAME            CHARACTER_DATA,
       VALID_SCALE           INTERVAL,
       SCALE_GRANULARITY     CHARACTER_DATA,
       VALID_PRECISION       INTERVAL,
       PRECISION_GRANULARITY CHARACTER_DATA,
       DISTRIBUTION          CHARACTER_DATA,
       GENERAL               CHARACTER_DATA,
       DEFAULT_EVENT         NONSTANDARD GENERAL INDETERMINATE TIMESTAMP,
       DEFAULT_STATE         NONSTANDARD GENERAL INDETERMINATE PERIOD,
       CONSTRAINT            TEMPORAL_SPEC_PRIMARY_KEY
                             PRIMARY_KEY (TABLE_NAME),
       CONSTRAINT            DISTRIBUTION_CHECK
                             CHECK (DISTRIBUTION IN ('STANDARD', 'NONSTANDARD'))
       CONSTRAINT            GENERAL_CHECK
                             CHECK (GENERAL IN ('NONGENERAL', 'GENERAL'))
       }
```

The precision and scale of the VALID_SCALE and VALID_PRECISION columns is the maximum supported by the implementation.

## 18.3 Section 21.3.?? SURROGATE base table

```
CREATE TABLE        SURROGATE {
       TABLE_NAME   CHARACTER_DATA,
       COLUMN_NAME  CHARACTER_DATA,
       CONSTRAINT   SURROGATE_PRIMARY_KEY
                    PRIMARY_KEY (TABLE_NAME, COLUMN_NAME)
       }
```

## 19 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 |
| | | property table non-existent | 009 |

## 20 History

Temporal databases have been an active research topic for at least fifteen years. During this time, several dozen temporal query languages have been proposed. In April, 1992 Richard Snodgrass circulated a white paper proposing that a temporal extension to SQL be produced by the research community. In parallel, the temporal database community organized the "ARPA/NSF International Workshop on an Infrastructure for Temporal Databases," which was held in Arlington, TX, in June, 1993. Discussions at that workshop indicated that there was substantial interest in a temporal extension to SQL-92. A general invitation was sent to the community, and about a dozen people volunteered to develop a language specification (over the next few months another half-dozen people joined the committee). The group corresponded via email from early July, 1993, submitting, debating, and refining proposals for the various portions of the language. In September, 1993, the first draft specification, accompanied by thirteen commentaries, was distributed to the committee. In December, 1993 a much enlarged draft, accompanied by some twenty-five commentaries, was distributed to the committee. A preliminary language specification appeared in the March, 1994 issue of *ACM SIGMOD Record*, and twenty-three commentaries were made available via anonymous FTP at `FTP.cs.arizona.edu`. A tutorial of the language appeared in the September, 1994 issue of *ACM SIGMOD Record*, and the final language specification and twenty-eight commentaries were also made available via anonymous FTP that month.

## Contributors

TSQL2 is remarkable, and perhaps unique, in that it was designed entirely via electronic mail, by a committee that never met physically (in fact, no one on the committee has met every other committee member).

The language design committee is quite broad, comprising members from database vendors, industrial research labs, industrial users, and academia. Committee members reside in seven countries on three continents. All committee members have published scholarly papers in the area of databases; for most, temporal databases is their primary research focus.

The TSQL2 Language Design Committee consists of Richard T. Snodgrass (chair), Department of Computer Science, University of Arizona, Tucson, `rts@cs.arizona.edu`; Ilsoo Ahn, AT&T Bell Laboratories, Columbus Ohio, `ahn@cbnmva.att.com`; Gad Ariav, Computer and Information Systems, Tel Aviv University, Israel, `ariavg@ccmail.gsm.uci.edu`; Don Batory, Department of Computer Sciences, University of Texas at Austin, `dsb@cs.utexas.edu`; James Clifford, Information Systems Department, New York University, `jcliffor@is-4.stern.nyu.edu`; Curtis E. Dyreson, Department of Computer Science, University of Arizona, Tucson, `curtis@cs.arizona.edu`; Christian S. Jensen, Department of Mathematics and Computer Science, Aalborg University, Denmark, `csj@iesd.auc.dk`; Ramez Elmasri, Computer Science and Engineering Department, University of Texas at Arlington, `elmasri@cse.uta.edu`; Fabio Grandi, University of Bologna, Italy, `fabio@deis64.cineca.it`; Wolfgang Käfer, Daimler Benz, Ulm, Germany, `kaefer%fuzi.uucp@germany.eu.net`; Nick Kline, Department of Computer Science, University of Arizona, Tucson, `kline@cs.arizona.edu`; Krishna Kulkarni, Tandem Computers, Cupertino, CA, `kulkarni_krishna@tandem.com`, Ting Y. Cliff Leung, Data Base Technology Institute, IBM, San Jose, CA, `cleung@almaden.ibm.com`; Nikos Lorentzos, Informatics Laboratory, Agricultural University of Athens, Greece, `eliop@isosun.ariadne-t.gr`; John F. Roddick, University of South Australia, The Levels, South Australia, `roddick@unisa.edu.au`; Arie Segev, University of California, Berkeley, CA, `segev@csr.lbl.gov`; Michael D. Soo, Department of Computer Science, University of Arizona, Tucson, `soo@cs.arizona.edu`; and Surynarayana M. Sripada, European Computer-Industry Research Centre, Munich, Germany, `spripada@ecrc.de`.

## Acknowledgements

Richard Snodgrass, Curtis Dyreson, Nick Kline, and Michael Soo were supported in part by National Science Foundation grants ISI-8902707 and ISI-9302244, IBM contract #1124 and a grant from the AT&T

Foundation. Christian S. Jensen was supported in part by the Danish Natural Science Research Council under grants 11–1089–1 SE and 11–0061–1 SE. John F. Roddick was supported in part by research grants from the University of South Australia and the Institute of Computer Systems Engineering and Assurance.

We appreciate the contributions of Petra Bayer, Suchen Hsu, Luis Hermosilla, Tomás Isakowitz, Raghu Ramakrishnam and Y. Mitsopoulos in writing several of the commentaries. We also appreciate the comments of Michael Böhlen, Shashi Gadia, Sushil Jajodia, Henry Kucera, Robert Marti, Nelson Mattos, Sham Navathe, Leo So and Abdullah Tansel. Finally, we thank Patrick P. Kalua, Angelo Montanari, Sunil S. Nair, Elisa Peressi, Barbara Pernici, Edward L. Robertson, Nanlal L. Sarda, Maria Rita Scala, Paolo Tibertio, Alexander Tuzhilin and Gene T.J. Wuu for their role in developing the consensus test suite of temporal database queries, which was very helpful in evaluating and refining TSQL2.