TSQL2 Language Specification*

Richard T. Snodgrass Ilsoo Ahn Gadi 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

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 modification and extension of SQL-92. The functionality of user-defined time support in SQL-92 is enhanced. This required replacing the `DATETIME` and `INTERVAL` types with alternative timestamp types. Section 6 discusses how legacy SQL-92 applications can be supported in TSQL2 without any code or SQL statement modifications.

The rest of the language, supporting temporal relations, is an upward-compatible extension of SQL-92.

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

*Correspondence may be directed to the chair of the TSQL2 Language Design Committee, Richard 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.

3 Definitions, notations, and conventions

This document 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 (addition is implementation defined!, and it is not stated which of seven possible definitions of second is used.) 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 extension of SQL-92.
- TSQL2 should allow the restructuring of relation instances 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 be optional.

  Relations that are not specified to be temporal should be considered to be snapshot relations. 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 relations, when temporal support is not desired. Similarly, it should be possible to derive a snapshot relation from a temporal relation.
- User-defined time support should include instants, intervals, and fixed and variable spans.

  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 highly varying uses of databases demand much more flexibility.
- It should be possible to derive temporal and non-temporal relations from underlying temporal and non-temporal relations.

Finally, we made ease of implementation a priority.

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

  In particular, the language should be implementable via a data model that employs interval-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.

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). An interval is modeled by the composition of two instant timestamps and the constraint that the instant timestamp that starts the interval equals or precedes (in the given scale) the instant timestamp that terminates the interval.

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 seconds, but does not indicate which of at least seven definitions of second is used.

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 interval in which the baseline clock uses UTC, 26 leap seconds were added. The baseline clock runs on UTC until one second before Midnight, July 1, 1993. 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 July 1, 1993, 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 replaced with more precise *instants*, *intervals*, and *spans* 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 a span (e.g., a precision of a millisecond). 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 having time-varying attributes.

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 `inception`, 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 know 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, span) 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 *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 maximal intervals. As an example, the Employee table with attributes Name, Salary and Manager could contain the tuple (Sam, 10000, LeeAnn). The temporal element timestamp would record the maximal (noncontiguous) intervals in which Sam made $10000 and had LeeAnn as his manager. Information about other values of Sam'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

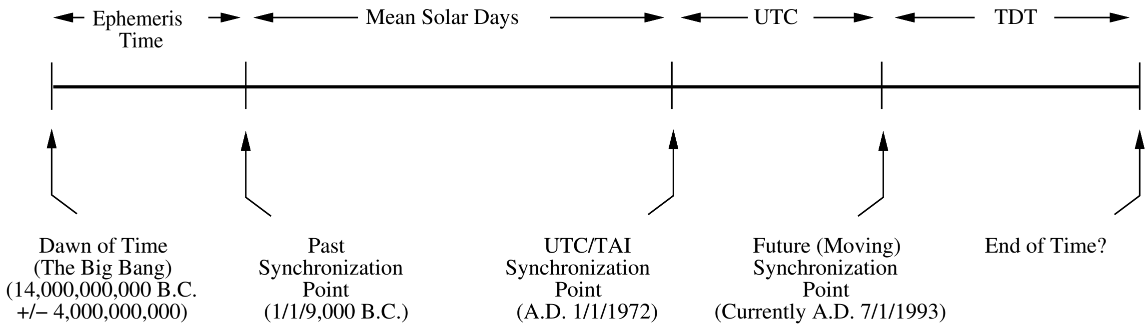

Figure 1: The time-line clock

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 (Sam, 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 interval 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 relations. 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 Sam 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 intervals when Sam made $10000, information which might be gathered from several underlying tuples specifying different managers.

Similarly, to determine when Sam had LeeAnn as his manager, independent of his salary, the table would be restructured on the Name and Manager columns. To determine when Sam 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 intervals or instants, respectively. Many queries refer to a continuous property, in which maximal intervals are relevant.

5.10 Temporal Selection

The valid-time timestamp of a table may participate in predicates in the `WHERE` clause by simply mentioning 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 relations, as well as valid-time relations, can be derived from underlying snapshot or valid-time relations. An optional `VALID` or `VALIDINTERSECT` 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 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

Some aspects of TSQL2 are pure extensions of SQL-92. The user-defined time in TSQL2 is a 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 `SQL-92` calendric system, along with either a preprocessor or a compatibility option implemented by the DBMS. In either case, it is possible to run legacy SQL-92 applications with no host language or embedded SQL changes required.

The defaults for the new clauses used to support valid-time relations 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 an interval-stamped (or instant-stamped, for event relations) 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 <nondelimiter token> is replaced, and a new non-terminal <calendar defined identifier> is added to define the format of calendric system defined identifiers. Calendric system defined identifiers are prefixed with an ampersand ("`&`").

```
  <nondelimiter token> ::=
          <regular identifier>
        | <key word>
        | <unsigned numeric literal>
        | <hex string literal>
•       | <instant literal>
•       | <interval literal>
•       | <span literal>

  <calendar defined identifier> ::=
•         <ampersand> <identifier>
```

The production for the non-terminal <reserved word> is changed. The reserved words `CURRENT`, `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`, `CURRENT_UTC_TIME`, `CURRENT_UTC_TIMESTAMP`, `DATE`, `DAY`, `EXTRACT`, `HOUR`, `LOCAL`, `MINUTE`, `MONTH`, `SECOND`, `TIME`, `TIMESTAMP`, `UTC`, `YEAR`, and `ZONE` are deleted.

The following productions are added for the non-terminal <reserved word>. To conserve space, we do not copy the existing reserved word definitions from the SQL92 document.

```
  <reserved word> ::=
•       | ALIGN
•       | CALENDRIC | CONTAINS | CREDIBILITY
•       | GENERAL
•       | INDETERMINATE | INSTANT
•       | MEETS
•       | NEW | NOBIND | NONSTANDARD
•       | PLAUSIBILITY | PRECEDES | PREVIOUS | PROPERTIES
•       | RISING
•       | SCALE | SNAPSHOT | SPAN | STATE | SURROGATE
•       | TRANSACTION
•       | VALID | VALIDINTERSECT
•       | WEIGHTED
```

Nineteen reserved words are deleted and 24 are added, a net addition of five reserved words.

10.2 Section 5.3 <literal>

The <datetime literal> non-terminal and its entire subtree are deleted. This is a calendar-specific language construct and definition of such would appear in a calendar.

The <interval literal> non-terminal and its entire subtree are deleted. The <interval literal> non-terminal which defined time durations is effectively replaced with a new non-terminal <span literal>.

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

```
<general literal> ::=
        <character string literal>
      | <national character string literal>
      | <bit string literal>
      | <hex string literal>
•     | <instant literal>
•     | <interval literal>
•     | <span literal>
```

The following productions are added.

```
<instant literal> ::=
•       [ <introducer> <character set specification> ]
             <vertical bar> <instant value> <vertical bar>
                  <calendric-property specification>

<interval literal> ::=
•       [ <introducer> <character set specification> ]
             <left bracket> <interval value>
                  {<right bracket> | <right paren>}
             <calendric-property specification>

<span literal> ::=
•       [ <introducer> <character set specification> ]
             <percent> <span value> <percent>
                  <calendric-property specification>

<instant value> ::=
•       !! See the syntax rules

<interval value> ::=
•       !! See the syntax rules

<span value> ::=
•       !! See the syntax rules
```

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

```
<format string> ::=
•       <quote> [ <background character> |
          field specification> ]...  <quote>

<background character> ::=
•       <character representation>

<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> ::=
•       !! See the syntax rules
```

Additional syntax rules:

1. An <instant value> is any sequence of characters not containing a single <vertical bar>. The format of an <instant value> is described by the current value of the *instant_input_format* property.
2. Within an <instant value> a <vertical bar> is represented by <vertical bar><vertical bar>.
3. An <interval value> is any sequence of characters not containing a single <right bracket> or a single <right paren>. The format of an <interval value> is described by the value of the *interval_input_format* property.
4. Within an <interval value> a single <right bracket> is represented as <right bracket><right bracket>, and a single <right paren> is represented as <right paren><right paren>.
5. A <span value> is any sequence of characters not containing a single <percent>. The format of a <span value> is described by the current value of the *span_input_format* property.
6. Within a <span value> a single <percent> is represented as <percent><percent>.
7. The data type of an <instant literal> is `INSTANT`.
8. The data type of an <interval literal> is `INTERVAL`.
9. The data type of a <span literal> is `SPAN`.
10. Interval literals ending with a <right bracket> are closed-closed intervals. Intervals ending with a <right paren> are closed-open intervals.
11. The non-terminal <calendric-property specification> is defined in Appendix 15.1.
12. If <calendric-property specification> contains an <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.

13. 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.

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

15. 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.

16. 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>.

17. Valid <field formatting specification>s are as follows.

    - `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.

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

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

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

Additional general rules:

1. The chronon denoted by an instant literal is assumed to be the first chronon represented by the instant string. This behavior may be changed with appropriate field names.

2. Interval literals are interpreted as follows. The beginning chronon of the interval is the first chronon contained in the interval, and the ending chronon of the interval is the last chronon contained in the interval. This behavior may be changed with appropriate field names.

3. Closed-closed intervals are closed on both ends (i.e., the interval includes both specified instants). Closed-open intervals do not contain their specified ending instant; they terminate one chronon before their ending instant.

10.3 Section 5.4 Names and identifiers

The following productions are added.

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

<property table name> ::=
• <table name>

Additional syntax rules:

1. The identifiers denoting calendric systems and property tables are implementation dependent.

11 Section 6 Scalar Expressions

11.1 Section 6.1 <data type>

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

<data type> ::=
<character string type> [`CHARACTER SET` <character set specification>]
| <national character string type>
| <bit string type>
| <numeric type>
• | <instant type>
• | <interval type>
• | <span type>
• | <surrogate type>

The <datetime type> non-terminal and its entire subtree are deleted. This is a calendar specific language construct.

The production for the <interval type> non-terminal is replaced, and new productions for the non-terminals <instant type> and <span type> are added as follows.

<instant type> ::=
• [<indeterminate data type>] `INSTANT` [<time precision and scale>]

<interval type> ::=
- [<indeterminate data type>] `INTERVAL` [<time precision and scale>]

<span type> ::=
- [<indeterminate data type>] `SPAN` [<time precision and scale>]

The following productions are added. <time precision> differs from <precision> in that a span is also permitted. <time scale> differs from <scale> in that int the former negative integers are also allowed.

<time precision and scale> ::=
- <left paren> <time precision> [<comma> <time scale>] <right paren>

<time precision> : : =
- <unsigned integer>
- | <span value>

<time scale> : : =
- <signed integer>
- | <span value>

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

Additional syntax rules:

1. If a <scale> is omitted, then zero is implicit. If a <precision> is omitted then an implementation-defined <precision> is implicit.
2. The default instant is determinate.
3. The default indeterminate instant is compact (not general).
4. The default distribution is standard.
5. 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.

Additional general rules:

1. A negative <scale> implies a granularity of 10 to the <scale> power.
2. The delimiting instants of an interval shall have the same precision and scale.
3. The permissible integer values for both the precision and the scale are implementation defined. The permissible span values are calendar-dependent, and cannot exceed the implementation capacities.
4. 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.
5. A special reserved word, `NEW` may be used when updating an attribute value of `SURROGATE` type. The new value is a previously unused value.
6. Values of type `SURROGATE` can only be compared with respect to identity.

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 15.1.
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 `INSTANT`, `INTERVAL`, or `SPAN`, 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 `INSTANT`, `INTERVAL`, or `SPAN`, 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 ] <correlation name>
               { <correlation> }... ]
•       | <derived table> [ [ AS ] <correlation name>
               { <correlation> }... ]
        | <joined table>
```

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

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

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

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

```
<partitioning unit> ::=
          INSTANT
        | INTERVAL
```

Additional syntax rules:

1. <coalescing attributes> 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 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.

4. If <partitioning unit> is not specified, then no partitioning is assumed.

11.4 Section 6.5 <set function specification>

The production for the non-terminal <set function specification> is changed to allow for aggregate functions defined by a calendric system.

```
<set function specification> ::=
          COUNT <left paren> <asterisk> <right paren>
        | <general set function>
•       | <calendar defined set function>
```

```
<calendar defined set function> ::=
•         <calendar defined identifier> <left paren>
              [ <set quantifier> ]
                  <value expression> <right paren>
```

We added an optional clause to the general set function production 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. A <calendar defined set function> is denoted by a <calendar defined identifier> prefixed with an <ampersand>.

2. Specifying `DISTINCT` in a <calendar defined set function> removes null values and duplicate values before computing the the aggregate function.

3. Specifying `ALL` in a <calendar defined set function> removes null values but does not remove duplicate values before computing the the aggregate function. `ALL` is the default option.

4. Let T be the data type of the values given as arguments to the <set function specification>.

5. If `SUM` is specified then T may not be `INSTANT` or `INTERVAL`.
6. Let DT be the data type of the <value expression>.
7. If `RISING` is specified, the data type of the result is an interval.
8. If `SUM` is specified, DT shall not be an instant or an interval.
9. If `AVG` is specified, DT shall not be an interval or a temporal element.
10. If `COUNT` is specified, `WEIGHTED` is not permitted.

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 interval such that the argument source T is monotonic increasing. If `WEIGHTED` is specified, then the largest interval 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 a span, in which case return the span 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 a span, in which case return the span with the maximal number of granules.

   (c) If `SUM` is present, if the type of A is a span, then return a span equal in length to the sum of the granules in T. Otherwise, the type of A must be a temporal element, 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 a span, then return a span 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.

11.5 Section 6.8 <datetime value function>

The <datetime value function> non-terminal and its entire subtree are deleted.

The following productions are added, including expressions evaluating to or taking as a parameter temporal expressions.

<instant value function> ::=
- `BEGIN` <left paren> <interval value expression> <right paren>
- | `END` <left paren> <interval value expression> <right paren>
- | `FIRST` <left paren> <instant value expression> <comma> <instant value expression> <right paren>
- | `FIRST` <left paren> <temporal element value expression> <right paren>
- | `FIRST` <left paren> <instant set value expression> <right paren>
- | `LAST` <left paren> <instant value expression> <comma> <instant value expression> <right paren>
- | `LAST` <left paren> <temporal element value expression> <right paren>
- | `LAST` <left paren> <instant set value expression> <right paren>
- | `NOBIND` <left paren> <instant literal> <right paren>
- | `NOBIND` <left paren> <column reference> <right paren>
- | <calendar defined identifier> [<left paren> <token> [<comma> <token>]... <right paren>]
- | `SCALE` <left paren> <instant value expression> <comma> <time scale> <right paren>
- | `ALIGN` <left paren> <instant value expression> <comma> <span value expression> <right paren>

```
<interval value function> ::=
•       INTERVAL <left paren> <instant value expression>
            <comma> <instant value expression>
            <right paren>
•     | INTERSECT <left paren>
            <interval value expression> <comma>
            <interval value expression> <right paren>
•     | NOBIND <left paren> <interval literal>
            <right paren>
•     | NOBIND <left paren> <column reference>
            <right paren>
•     | <calendar defined identifier> [ <left paren>
            <token> [ <comma> <token> ]...
            <right paren> ]
•     | SCALE <left paren> <interval value expression>
            <comma> <time scale> <right paren>
•     | ALIGN <left paren> <interval value expression>
            <comma> <span value expression>
            <right paren>
•     | FIRST <left paren>
            <temporal element value expression>
            <right paren>
•     | LAST <left paren>
            <temporal element value expression>
            <right paren>
```

```
<span value function> ::=
•       SPAN <left paren> <interval value expression>
            <right paren>
•     | SPAN <left paren>
            <temporal element value expression>
            <right paren>
•     | ABSOLUTE <left paren> <span value expression>
            <right paren>
•     | NOBIND <left paren> <span literal> <right paren>
•     | NOBIND <left paren> <column reference>
            <right paren>
•     | <calendar defined identifier> [ <left paren>
            <token>  [ <comma> <token> ]...
            <right paren> ]
•     | SCALE <left paren> <span value expression>
            <comma> <time scale> <right paren>
•     | ALIGN <left paren> <span value expression>
            <comma> <span value expression>
            <right paren>
```

```
<temporal element value function> ::=
•       INTERSECT <left paren>
            <temporal element value expression>
            <comma>
            <temporal element value expression>
            <right paren>
```

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

```
<instant set value function> ::=
•       INTERSECT <left paren>
            <instant set value expression> <comma>
        <instant set value expression> <right paren>
```

Additional syntax rules:

1. <calendar defined identifier> must be a function defined via the currently active calendric system.
2. The number and types of the tokens in the parameter list of a <calendar defined identifier> function must agree in number and type with the definition of the function in the calendric system. A compilation error will be generated if any type errors are detected.

Additional general rules:

1. `FIRST` (`LAST`) extracts the first (last) maximal interval from the temporal element.
2. `FIRST` (`LAST`) extracts the first (last) instant from the instant set.
3. Intersection of temporal elements is set intersection.
4. Local invocation of a scale or align function overrides the global default.
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 6.11 <value expression>

The production for the non-terminal <value expression> is replaced with the following. Value expressions are augmented to include expressions evaluating to temporal elements. Value expressions are augmented to include expressions evaluating to instant sets.

```
<value expression> ::=
        <numeric value expression>
      | <string value expression>
•     | <instant value expression>
•     | <interval value expression>
•     | <span value expression>
•     | <temporal element value expression>
•     | <instant set value expression>
```

11.7 Section 6.12 <numeric value expression>

The non-terminal <extract expression> and its entire subtree is deleted, and the production for the non-terminal <numeric primary> is changed to the following.

```
<numeric primary> ::=
            <unsigned value specification>
          | <column reference>
          | <set function specification>
          | <scalar subquery>
          | <case expression>
          | <left paren> <numeric value expression>
                <right paren>
•         | <calendar defined identifier> [ <left paren>
                <token> [ <comma> <token> ]...
                <right paren> ]
          | <numeric cast specification>
          | <position expression>
          | <length expression>
```

Additional syntax rules:

1. <calendar defined identifier> must be an <identifier> representing a calendar defined function that returns a numeric value.
2. <calendar defined identifier> can be used to define field extraction functions for temporal values.
3. A calendar of the currently active calendric system is assumed to define the <calendar defined identifier>. If no `SET CALENDRIC SYSTEM` command has been entered during the session then a calendar of the implementation defined default calendric system is assumed to define the <calendar defined identifier>. Any other situation generates a compilation error.
4. The number and types of the tokens in the parameter list of a <calendar defined identifier> function must agree in number and type with the definition of the function in the calendric system. Any other situation generates a compilation error.

11.8 Section 6.13 <string value expression>

The production for the non-terminal <character primary> is changed to the following.

```
<character primary> ::=
            <unsigned value specification>
          | <column reference>
          | <set function specification>
          | <scalar subquery>
          | <case expression>
          | <substring>
          | <fold>
          | <left paren> <character value expression>
                <right paren>
          | <character cast expression>
          | <character translation>
          | <form-of-use conversion>
•         | <calendar defined identifier> [ <left paren>
                <token>  [ <comma> <token> ]...
                <right paren> ]
```

Additional syntax rules:

1. <calendar defined identifier> must be an <identifier> representing a calendar defined function that returns a string value.
2. A <calendar defined identifier> can be used to define field extraction functions for temporal values.
3. A calendar of the currently active calendric system is assumed to define the <calendar defined identifier>. If no `SET CALENDRIC SYSTEM` command has been entered during the session then a calendar of the implementation defined default calendric system is assumed to define the <calendar defined identifier>. Any other situation generates a compilation error.
4. The number and types of the tokens in the parameter list of a <calendar defined identifier> function must agree in number and type with the definition of the function in the calendar. Any other situation generates a compilation error.

11.9 Section 6.14 <datetime value expression>

The non-terminal <datetime value expression> and its entire subtree are replaced with the following productions. The production for the non-terminal <instant primary> is augmented to also include references to tables themselves. Also, expressions evaluating to instant sets are added.

```
<instant value expression> ::=
•           <instant term>
•         | <span value expression> <plus sign>
                <instant value expression>
•         | <instant value expression> {<plus sign> |
                <minus sign>} <span value expression>

<instant term> ::=
•           <instant factor>

<instant factor> ::=
•           <instant primary>

<instant primary> ::=
•           <instant literal>
•         | <column reference>
•         | <scalar subquery>
•         | <case expression>
•         | <instant value function>
•         | <left paren> <instant value expression>
                <right paren>
•         | <cast specification>
•         | <table name>
•         | <correlation name>
•         | VALID <lparen> { <table name> |
                <correlation name> } <right paren>
```

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

<interval primary> ::=
- <interval literal>
- | <column reference>
- | <scalar subquery>
- | <case expression>
- | <interval value function>
- | <cast specification>
- | <table name>
- | <correlation name>
- | `VALID` <lparen> { <table name> | <correlation name> } <right paren>

<span value expression> ::=
- <span term>
- | <span value expression> {<plus sign> | <minus sign>} <span term>
- | <instant value expression> <minus sign> <instant term>
- | <left paren> <instant value expression> <minus sign> <instant term> <right paren>

<span term> ::=
- <span factor>
- | <span term> {<asterisk> | <solidus>} <factor>
- | <factor> <asterisk> <span term>

<span factor> ::=
- [<minus sign>] <span primary>

<span primary> ::=
- <span literal>
- | <column reference>
- | <scalar subquery>
- | <case expression>
- | <left paren> <span value expression> <right paren>
- | <cast specification>
- | <span value function>

<temporal element value expression> ::=
- <temporal element value term>
- | <temporal element value expression> { '+' | '-' } <temporal element value term>

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

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

<temporal element value primary> ::=
- <table name>
- | <correlation name>
- | `VALID` <lparen> { <table name> | <correlation name> } <right paren>
- | `TRANSACTION` <lparen> { <table name> | <correlation name> } <right paren>
- | <temporal element value function>

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

<instant set value primary> ::=
- <table name>
- | <correlation name>
- | `VALID` <lparen> { <table name> | <correlation name> } <right paren>
- | <instant set value function>

Additional syntax rules:

1. The data type of an <instant primary> is `INSTANT`.
2. The data type of an <interval primary> is `INTERVAL`.
3. The data type of a <span primary> is `SPAN`.
4. The data type of an <instant value expression> is `INSTANT`.
5. The data type of an <interval value expression> is `INTERVAL`.
6. The data type of a <span value expression> is `SPAN`.
7. Table 1 lists the arithmetic expressions involving time values that are valid.

Additional general rules:

1. Let T be the <table> or <correlation name>.
2. `VALID` is only permitted on valid-time and bitemporal tables.
3. `TRANSACTION` is only permitted on transaction-time and bitemporal tables.
4. Case:
   (a) If T is a valid-time or bitemporal table, then the <table name> or <correlation name> alone is equivalent to `VALID(<name>)`.
   (b) If T is a transaction-time table, then the <table name> or <correlation name> alone is equivalent to `TRANSACTION(<name>)`.

Operand 1	*Operator*	*Operand 2*	*Yields*
	-	*span*	*span*
span	+	*span*	*span*
span	-	*span*	*span*
instant	+	*span*	*instant*
instant	-	*span*	*instant*
span	+	*instant*	*instant*
instant	-	*instant*	*span*
span	*	*numeric*	*span*
numeric	*	*span*	*span*
span	/	*numeric*	*span*
span	/	*span*	*numeric*
span	+	*interval*	*interval*
interval	+	*span*	*interval*
interval	-	*span*	*interval*
element	+	*element*	*element*
element	-	*element*	*element*
event set	+	*event set*	*event set*
event set	+	*event set*	*event set*

Table 1: Valid Arithmetic Expressions and Results.

5. If T is a non-partitioned correlation variable associated with a valid-time state table, then `VALID` is a temporal element. If T is a correlation variable partitioned by `INTERVAL`, then `VALID` is an interval. If T is a non-partitioned correlation variable associated with a valid-time event table, then `VALID` is an instant set. If T is a correlation variable partitioned by `INSTANT`, then `VALID` is an instant.

6. Operands are coerced to the global scale specified in the last `SET DEFAULT SCALE` command prior to the operation.

7. The range and scale of intermediate results are the maximum allowed by the implementation.

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

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

11.10 Section 6.15 <interval value expression>

The non-terminal <interval value expression> and its subtree are deleted. It is effectively replaced by the new non-terminal <span value expression>.

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>
[<group by clause>]
[<having clause>]

The following production is added.

<valid clause> `::=`
• { `VALID` | `VALIDINTERSECT` }
{ <element value expression>
| <interval value expression> |
<instant value expression> }
[`WITH CREDIBILITY` <integer>]

Additional general rules:

1. `VALIDINTERSECT` T is equivalent to `VALID INTERSECT(`T, `INTERSECT(`C_1`,` ... `INTERSECT(`C_{n-1}`,` C_n`)`, where C_i are the correlation variables (or table names) mentioned in the `SELECT` clause.

2. The default `VALID` clause is `VALIDINTERSECT INTERVAL(|beginning now|, UNBIND(|now forever|))`.

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

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.

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).
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> ] }
•       | {    {    TRANSACTION <left paren>
                <table reference> <right paren>
•                  | VALID <left paren>
                <table reference> <right paren>
•                  | <column reference> }
•               [ USING { <span> | INSTANT } ]
                [ LEADING <span> ] [ TRAILING <span> ] }
```

Additional syntax rules:

1. If the using clause, or the leading clause or the trailing clause is present, and neither `VALID` or `TRANSACTION` is, then the type of the <column reference> must be a timestamp.
2. `VALID` or `TRANSACTION` may only be present once in a <group by clause>.

Additional general rules:

1. If the type of the <column reference> is a timestamp, or `TRANSACTION` or `VALID` is present, then

   Case:

   (a) If the using clause is not present, then the default is `INSTANT`.

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

2. If any or all of the using, trailing or leading clauses are present, or `VALID` or `TRANSACTION` is present, or the type of the <column reference> is a timestamp, then partition the table value the following way.

   (a) Partition the time-line of the <table reference> or the <column reference> according to the using clause. If the using clause contains `INSTANT`, partition the time-line into separate partitions for each granule. If the partition is a span, partitioning is according to the current calendar.

   (b) For each partition, determine the trailing and leading intervals induced by their spans. This may be calendar dependent. The leading interval, if present, will contain the granule next to the first granule of the partition. Similarly, the first granule of the trailing interval will be the first granule after the partition. The partition will not share granules with either induced interval. Let I be the interval containing the three intervals.

   (c) Let T be the result of the preceding <from clause>, or the result of previous subsets induced by the <group by clause>. For each partition of the time-line, and each group in T, associate the tuples from T which overlap with the interval I with the current partition.

12.5 Section 7.8 <having clause>

Additional general rules:

1. Let T be one of the clauses in the <group by clause>.
2. If T included `TRANSACTION` or `VALID`, or T was followed by a using clause, leading clause, or trailing clause, and T is present in the <having clause>, then the using clause must not contain a span larger than a granule, and the leading and trailing clauses must be zero length.

12.6 Section 7.9 <query specification>

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

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

Additional general rules:

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

13 Section 8 Predicates

13.1 Section 8.1 <predicate>

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

```
<predicate> ::=
          <comparison predicate>
        | <between predicate>
        | <in predicate>
        | <like predicate>
        | <null predicate>
        | <quantified comparison predicate>
        | <exists predicate>
```

| <unique predicate>
| <match predicate>
• | <precedes predicate>
• | <meets predicate>
• | <overlaps predicate>
• | <contains predicate>

13.2 Section 8.2 <comparison predicate>

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

1. The <less than operator>, <greater than operator>, and <equals operator> are valid for span 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 `INTERVAL`, `INSTANT`, or `SPAN` 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 left as is though its semantics have changed.)

<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 SQL92 proposal 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 `INTERVAL` or `INSTANT`.
4. T_1 and T_2 shall be comparable as defined in Table 2.
5. Any comparison involving the `INTERVAL` or `INSTANT` data types not listed in Table 2 is disallowed.

Operand 1	*Operator*	*Operand 2*
span	=	*span*
span	<	*span*
span	>	*span*
instant/interval	=	*instant/interval*
instant/interval	PRECEDES	*instant/interval*
instant/interval	OVERLAPS	*instant/interval*
instant/interval	CONTAINS	*instant/interval*
instant/interval	MEETS	*instant/interval*

Table 2: Permitted Comparison Operators

Additional general rules:

1. Operands are coerced to the global scale specified in the last `SET DEFAULT SCALE` command prior to the operation.

13.5 Section 10.1 <interval qualifier>

The non-terminal <interval qualifier> and its subtree are deleted. This is a calendar-specific language construct and definition of such would appear in a calendar.

14 Section 11 Schema definition and manipulation

14.1 Section 11.3 <table definition>

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

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

Two productions are added.

<temporal definition> ::=
• <valid definition> [`AND TRANSACTION`]
• | `AS TRANSACTION`

<valid definition> ::=
• `AS` [`VALID`] { `STATE` | `EVENT` }
[<time precision and scale>] <default clause>

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 intervals. The precision and scale of the intervals 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.

14.2 Section 11.5 <default clause>

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

<default clause> ::=
    <literal>
• | <instant value function>
• | <interval value function>
• | <span value function>
  | `USER`
  | `SYSTEM USER`
  | `NULL`

Additional syntax rules:

1. If <instant value function>, <interval value function>, or <span 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 `INSTANT`, `INTERVAL`, or `SPAN` then `USER` and `SYSTEM USER` may not be specified.

4. If T is `INSTANT` then either a <literal> representing an <instant literal> or an <instant 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 `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.

6. If T is `SPAN` then either a <literal> representing a <span literal> or a <span 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.

14.3 Section 11.10 <alter table statement>

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

<alter table statement> ::=
• | <add valid definition>
• | <drop valid definition>
• | <replace valid definition>
• | <add transaction definition>
• | <delete transaction definition>

The following productions are added.

<add valid definition> ::=
• `ADD` [`VALID`] { `STATE` | `EVENT` }
    [<time precision and scale>]

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

<delete transaction definition> ::=
• `DROP TRANSACTION`

Additional syntax rules:

1. Let T be the table identified in the containing <alter table statement>.

2. T shall be a snapshot table or a transaction-time table.

<drop valid definition> ::=
• `DROP VALID`

Additional syntax rules:

1. Let T be the table identified in the containing <alter table statement>.
2. T shall be a valid-time or bitemporal table.

Additional general rules:

1. T is converted to a snapshot or transaction-time table.
2. If T is an interval table, it is converted to a snapshot table with contents

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

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

   `SELECT SNAPSHOT * FROM T`

<replace valid definition> ::=
- `REPLACE` [`VALID`] { `STATE` | `EVENT` } <time precision and scale>]

Additional syntax rules:

1. Let T be the table identified in the containing <alter table X1statement>.
2. T shall be a valid-time table.

Additional general rules:

1. If T was an state table and <valid definition> specifies interval, 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.
2. 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 intervals to a set of instants, equivalently,

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

3. 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.
4. If T was an event table and <valid definition> specifies interval, then the timestamp of each tuple in T is converted from a set of instants to a set of intervals, equivalently,

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

15 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> ::=
- `SET CALENDRIC SYSTEM WITH` <calendric spec>

15.1 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 credibility and plausibility defaults.

<SQL session statement> ::=
<SQL set identifier statement>
| <set constraints mode statement>
| <set transaction statement>
- | <set properties statement>
- | <set scale statement>
- | <set align statement>
- | <set credibility statement>
- | <set plausibility statement>

<set properties statement> ::=
- `SET PROPERTIES WITH` <property spec>

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

<set align statement> ::=
- `SET ALIGN` <time scale>

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

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

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

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

<property spec> ::=

- PREVIOUS
- | DEFAULT
- | <property table name>
- | <table value expression>

<set credibility statement> ::=

- SET CREDIBILITY { <integer> | AS DEFAULT }

<set plausibility statement> ::=

- SET PLAUSIBILITY { <integer> | AS DEFAULT }

Additional syntax rules:

1. In a sequence of SQL statements, the last calendric system specified in a `SET CALENDRIC SYSTEM` command remains in effect until a new `SET CALENDRIC SYSTEM` command is entered.
2. A `SET CALENDRIC SYSTEM WITH DEFAULT` statement reactivates the implementation defined default calendric system.
3. The non-terminal <calendric system name> must be an <identifier> naming a calendric system.
4. The non-terminal <property table name> is the name of a property table defining properties of the named calendric system.
5. The most recent invocation of a <set scale statement> or a <set align statement> takes precedence.
6. If both the <set scale statement> and the <set align statement> are omitted, then the default scale is assumed.
7. Specifying `SET PROPERTIES WITH PREVIOUS` causes the previous set of active properties to be reactivated.
8. Specifying `SET PROPERTIES WITH DEFAULT` causes the implementation defined set of default properties to be activated.
9. The non-terminal <table value expression> enumerates the rows of a property table.
10. 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. (The reserved word `VARCHAR` indicates a varying length character string.)

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

11. The most recent invocation of a <set credibility statement> or a <set plausibility statement> takes precedence.
12. If both the <set credibility statement> and the <set plausibility statement> are omitted, then the defaults, 100 and 100, respectively, are assumed.

16 Section 13 Data manipulation

16.1 Section 13.3 <fetch statement>

<fetch statement> ::=

FETCH [[<fetch orientation>] FROM]

- <cursor name> [INTO <fetch target list>]
- [INTO VALID [INTERVAL] <fetch target list>]

Additional syntax rules:

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

Additional general rules:

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

16.2 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>.

16.3 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.

```
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'))
```

Figure 2: The `TABLES` Base Table

<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 `INTERVAL(%now%, UNBIND(%now%))` is assumed.

16.4 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`.

16.5 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 `INTERVAL(|now|, |until changed|)`.

16.6 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 `INTERVAL(|now|, |until changed|)`.

17 Section 16 Session Management

17.1 Section 16.5 <set local time zone statement>

The non-terminal <set local time zone statement> and its entire subtree are deleted. Local time displacement is calendar specific.

18 Section 21 Information Schema and Definition Schema

18.1 Section 21.3.8 TABLES base table

See Figure 2.

18.2 Section 21.3.x TEMPORAL_SPEC base table

See Figure 3.

18.3 Section 21.3.y SURROGATE base table

See Figure 4.

```
CREATE TABLE        TEMPORAL_SPEC {
       TABLE_NAME    CHARACTER_DATA,
       TEMPORAL_TYPE CHARACTER_DATA,
       SCALE         SPAN,
       PRECISION     SPAN,
       DISTRIBUTION  CHARACTER_DATA,
       GENERAL       CHARACTER_DATA,
       DEFAULT_EVENT NONSTANDARD GENERAL INDETERMINATE INSTANT,
       DEFAULT_STATE NONSTANDARD GENERAL INDETERMINATE INTERVAL,
       CONSTRAINT    TEMPORAL_SPEC_PRIMARY_KEY
                     PRIMARY_KEY (TABLE_NAME),
       CONSTRAINT    TEMPORAL_TYPE_CHECK
                     CHECK (TEMPORAL_TYPE IN ('VALID STATE', 'VALID EVENT',
                         'TRANSACTION', 'BITEMPORAL STATE', 'BITEMPORAL EVENT'))
       CONSTRAINT    DISTRIBUTION_CHECK
                     CHECK (DISTRIBUTION IN ('STANDARD', 'NONSTANDARD'))
       CONSTRAINT    GENERAL_CHECK
                     CHECK (GENERAL IN ('NONGENERAL', 'GENERAL'))
       }
```

Figure 3: The `TEMPORAL_SPEC` Table

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

Figure 4: The `SURROGATE` Table

19 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.

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, those on the committee have never met everyone else individually).

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 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`; Gadi 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 Dept., 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, Afdeling for Matematik Og Datalogi, Aalborg Universitetscenter, 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.