PART IV

LANGUAGE SPECIFICATION

28

LANGUAGE SYNTAX

The organization of this section follows that of the SQL-92 standard. The syntax is listed under corresponding section numbers in the SQL-92 standard. 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 standard, and that a copy of the standard is available for reference.

29

SECTION 5 LEXICAL ELEMENTS

29.1 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 standard.

<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`

29.2 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>

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

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

<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>

<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 precedes 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 consistent 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 time zone 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, depending 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 consistent 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 time zone 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> is defined in Section 35.1.

13. If <calendric-property> 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> 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.*

29.3 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`.

30

SECTION 6 SCALAR EXPRESSIONS

30.1 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.

30.2 SECTION 6.2 <VALUE 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>]

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

Additional syntax rules:

1. The non-terminal <calendric-property> is defined in Section 35.1.
2. If <calendric-property> is specified then <parameter name> must have the data type <character string type>. Similar remarks apply to <embedded variable name>.
3. If <calendric-property> 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>.
4. If <calendric-property> 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> 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.

30.3 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 ultimate 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.

30.4 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 multiplied 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 multiplied 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. If the type of A is 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. If the type of A is period, then the beginning instant of the resulting period is the average of the beginning instants of the periods in T, and the terminating instant of the resulting period is the average of the terminating instants of the periods in T.

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.*

30.5 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.

30.6 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. Table 30.1 showing the 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.
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>.

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

<data type> of *SD*	<data type> of *TD* YM	DT	P	TE	IS	G
EN	M	M	N	N	N	N
AN	N	N	N	N	N	N
C	Y	Y	Y	Y	Y	N
B	N	N	N	N	N	N
D	N	N	Y	Y	Y	Y
T	N	N	Y	Y	Y	Y
TS	N	N	Y	Y	Y	Y
YM	Y	N	N	N	N	Y
DT	N	Y	N	N	N	Y
P	M	M	Y	Y	N	Y
TE	N	N	N	Y	Y	Y
IS	N	N	N	Y	Y	Y

Table 30.1 Cast data conversions

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*.

30.7 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>

30.8 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 30.2 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 time zone 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 left-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 <datetime term>, as well as the <period qualifier> are supplied by the `SQL92` calendar, then the semantics are as follows. (Original Rule 6 goes here.)

30.9 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 <datetime term>, as well as the <period qualifier> are supplied by the `SQL92` calendar, then the semantics are as follows. (Original Rule 6 goes here.)

30.10 SECTION 6.16 <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.

30.11 SECTION 6.17 <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 30.2 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.*

30.12 SECTION 6.18 <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> <right paren>
- | `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.

30.13 SECTION 6.19 <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).

30.14 SECTION 6.20 <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 value 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>.

30.15 SECTION 6.21 <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).

30.16 SECTION 6.22 <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.

31

SECTION 7 QUERY EXPRESSIONS

31.1 SECTION 7.1 <ROW VALUE CONSTRUCTOR>

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> }
```

31.2 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.

31.3 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.

31.4 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>
•       | <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 valid clause.

3. If the type of the <column reference> in a <temporal partition> clause is a timestamp, or `VALID` is present, then

   (a) If the using clause is not present, then the default is `INSTANT` for <partition expression>'s that contain `VALID`, and `PERIOD 'All of time'` <time granularity> for <column references> in a <temporal partition>. The default granularity is the granularity of the valid clause. 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 is 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 *tentatively associated* with the sequence of granules from L to G, inclusive.

(b) The using clause specifies how many consecutive granules ($\{ g_1, \ldots, g_n \}$) 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 $\{ g_1, \ldots, g_n \}$.

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

31.5 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>.

31.6 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>
```

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

```
<select list> ::=
        <column list>
      | <asterisk>
•     | <asterisk> <asterisk>
      | <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.

32

SECTION 8 PREDICATES

32.1 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>
```

32.2 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.

32.3 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 32.1.

32.4 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 32.1.
5. Any comparison involving the `PERIOD` or *datetime* data types not listed in Table 32.1 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 32.1 Permitted set of comparison operators

33

SECTION 10 ADDITIONAL COMMON ELEMENTS

33.1 SECTION 10.1 <INTERVAL QUALIFIER>

This is significantly generalized to allow implementation-defined granularities. The <non-second datetime 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.

34

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>

34.1 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 timestamped 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).

34.2 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.

34.3 SECTION 11.5 <DEFAULT CLAUSE>

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

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

Additional syntax rules:

1. 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.

34.4 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>
- | <drop 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.

34.5 SECTION 11.38 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.

35

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.

35.1 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> ::=
•         [ <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.*

36

SECTION 13 DATA MANIPULATION

36.1 SECTION 13.3 <FETCH STATEMENT>

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

Additional syntax rules:

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

Additional general rules:

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

statement, and exactly one <target specification> is an event table is being evaluated.

36.2 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>.

36.3 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.

36.4 SECTION 13.8 <INSERT STATEMENT>

A <valid value> is added.

```
<insert columns and source> ::=
        [ <left paren> <insert column list> <right paren> ]
•           [ <valid value> ]
            <query expression>
        | DEFAULT VALUES
```

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

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

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

Additional general rules:

1. `NEW` is permitted only when the <data type> of the corresponding column is `SURROGATE`.
2. 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.

36.5 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'))`.

36.6 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'))`.

37

SECTION 21 INFORMATION SCHEMA AND DEFINITION SCHEMA

37.1 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.