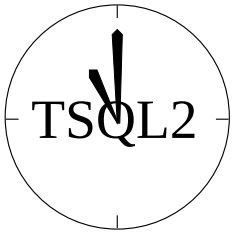

Now in TSQL2

September 26, 1994

A TSQL2 Commentary

The TSQL2 Language Design Committee

Title	Now in TSQL2
Primary Author(s)	James Clifford, Curtis Dyreson, Richard T. Snodgrass, Tomás Isakowitz and Christian S. Jensen
Publication History	April 1993. TempIS Technical Report #42. September 1994. TSQL2 Commentary.

TSQL2 Language Design Committee

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

Abstract

"Now" is a distinguished timestamp value used by many temporal data model proposals. In this paper, we propose a new kind of event, a *now-relative event*, that more accurately captures the semantics of "now." We discuss query language constructs, representation, and query processing strategies for such values. We demonstrate that these values incur no storage overhead and nominal additional query execution cost. The related concepts of "infinite future" and "infinite past" are also considered.

1 Introduction

Now is an English noun meaning "at the present time" [18]. *Now* is also a distinguished timestamp value in many temporal data model proposals. In this commentary, we give precise, but informal, semantics for this familiar term and propose representations, and query language constructs for supporting *now* in TSQL2. We also explore the related concepts of "infinite future" and "infinite past."

In general, we treat *now* as a variable that is assigned a specific time during query or update evaluation. The time that is assigned to the variable depends upon when the query or update is evaluated. We call the time assigned to the variable *now* the *reference time* as it is specific to the reference frame of the observer; in the case of *now* the observer is the query (or update). Now appears in SQL-92 though the reserved words `CURRENT_DATE`, `CURRENT_TIME`, and `CURRENT_TIMESTAMP`.

This discussion of *now* adds to the set of temporal values and types developed elsewhere [7, 17]. In particular, *now* is a distinguished kind of *datetime*, rather than a *interval* or *period*. A datetime is a fixed point on an underlying time-line whereas a period is an anchored segment of the time-line demarcated by two datetimes. An interval is the duration between two datetimes, an unanchored segment of the time-line. Two distinguished datetimes currently exist: *beginning*, which is the earliest time on the underlying time-line (valid or user-defined time), and *forever*, which is the latest time.

2 *Now* in User-defined and Valid Time

A common use of *now* is to indicate that a fact is valid until the current time [1, 2, 8, 9, 12, 13, 19, 23]. For example, suppose that Jane began working as a faculty member for State University on 06/01/94. Figure 1 shows the relevant tuple from the university's employment history (the `FACULTY1` relation). Jane started working as an Assistant professor at State University on 06/01/94, as indicated by the "valid time" attribute (for the examples in this paper we assume a timestamp granularity of one day). The variable *now*, appearing as the terminating datetime in the valid-time period for Jane's employment tuple, represents a currently unknown future time when Jane will stop working for State University. The result of a query that requests the current faculty members will include Jane.

The informal semantics of this value is that Jane is a faculty member until we learn otherwise. As the current time inexorably advances, the interpretation of *now* also changes to reflect the new current time. Some authors have called this concept "until changed" instead of "now" [21, 22], but the semantics is the same.

Other data models use *forever* or ∞ as the terminating period datetime, as shown in Figure 2 [3, 15, 16, 20]. Forever is the largest representable timestamp value, that is, the one furthest into the

FACULTY1		
NAME	*RANK*	*VALID-TIME*
Jane	Assistant	'[06/01/94 - *now*]'

Figure 1: Jane's employment tuple

FACULTY2		
NAME	*RANK*	*VALID-TIME*
Jane	Assistant	'[06/01/94 - forever]'

Figure 2: Jane's employment tuple with a large upper bound

future. This value admits that we do not know when Jane will depart the company, and so assumes that she will be working forever.

One limitation of using forever is that it is overly optimistic: forever is a long time into the future! In SQL and in IBM's DB2, forever is about 8,000 years from the present [5, 11]; in our more liberal proposal, it is approximately 18 *billion* years from the present time [6]. Hence, to assert that Jane will be employed until forever is most assuredly incorrect (others have also noted that a terminating time of ∞ or *forever*, has erroneous implications for the future [12]). A related limitation is that when Jane departs from the company, forever must be revised with the date of her departure; but the revised date will be an entirely separate time, unrelated to forever.

An alternative way to view this problem is that there is a difference between the *actual* and *expected* times of a fact. On a day-to-day basis, we expect Jane to remain employed. A database that uses *forever* as the terminating time of her employment tuple (very optimistically) records her expected employment, while a database that uses *now* records only her actual employment, the time she has worked to the current time.

2.1 Why use *Now*?

Suppose that instead of using the variable *now* as the terminating time in the tuple in Figure 1, we use a ground time, i.e., a particular date. Then as time advances and Jane remains an Assistant professor at State University, the terminating time on Jane's tuple must be updated each day to record when she worked. While this representation is faithful to our knowledge at any point in time, it is it is unrealistic to assume that the terminating time will be continuously updated as time advances. It is also unclear who should do the updating, as the database has no indication of which timestamp values are stable and which are continuously changing. For these reasons, it is more convenient to use the variable *now*.

2.2 Now-relative Datetimes

In this section we introduce a new kind of datetime, called a *now-relative datetime*. A now-relative datetime is a datetime that is located at a given offset from, or relative to, *now*, or the *reference time*. Now-relative datetimes are a proper subset of the general notion of a *parameterized datetime*. We show

FACULTY3		
NAME	*RANK*	*VALID-TIME*
Jane	Assistant	'[06/01/94 - (*now* + 3 days)]'

Figure 3: Using a now-relative datetime

below that now-relative datetimes are very useful.

The terminating time in Jane's employment tuple shown in Figure 1 is a now-relative datetime. For this datetime, the offset is a zero-length interval. By using now-relative datetimes, we can more accurately record our "actual" knowledge of Jane's employment with State University.

As an example, assume that all changes to the faculty database are made 3 days prior to when they take effect, then Jane's employment should extend from when she was hired to 3 days after *now* as shown in Figure 3. Here the terminating timestamp value is a rather complex datetime. It is an expression involving the variable *now* and a *interval*, in this case, 3 days, indicating the punctuality of updates. We recommend support for only those now-relative datetimes that indicate an offset from *now*, e.g., support for the addition operator. Now-relative datetimes involving multiplication (or division), such as `DATE '2*now'`, are not included in this proposal.

An interval is the duration between two datetimes, an unanchored segment of the time-line.

The processing of a now-relative datetime is quite interesting. First the variable *now* is bound to the reference time. Next, the arithmetic involving the interval (if any) is performed. In essence, the processing of a now-relative datetime is a non-relative datetime, calculated by substituting the reference time for *now* and subtracting (or adding) the interval. Finally the resulting tuple is used as expected in the query. For example, consider the processing of the tuple in Figure 3 on 07/09/94, e.g.,

$$< Jane, Assistant,' [06/01/94 - now + 3days]' > .$$

First *now* is bound to the reference time, 07/09/94. Next the interval arithmetic is performed:

$$07/09/94 + 3\ days\ =\ 07/12/94,$$

resulting in the tuple

$$< Jane, Assistant,' [06/01/94 - 07/12/94'] > .$$

The resulting tuple is then used in the rest of the query. The variable *now* is *always* ground *prior* to its use in a query. This is because the TSQL2 semantics is based upon tuples without variables, (especially, the semantics of arithmetic operations). We do not currently propose extending the semantics to support "unground" tuples or values.

2.3 An Aside: Forever and Beginning

The symbol `forever` used in a tuple has the following interpretation: *forever* $= \infty$. Here, ∞ is a special time in the temporal universe that is greater than any other time in that universe. As a consequence, we point out that the symbol `forever` is in fact not a variable, but a constant. The special symbol `beginning` is treated similarly (as $-\infty$).

FACULTY4		
NAME	*RANK*	*VALID-TIME*
Jane	Assistant	'[06/01/94 - 06/13/94]'

Figure 4: Executing the insert with |now|

3 *Now* at Run-time

Now is always bound to the current reference time when used in a query. There is one important exception to this maxim, the nobind() function discussed below. First consider the query given below.

```
SELECT NAME, RANK
FROM FACULTY
WHERE DATE 'now' OVERLAPS VALID(FACULTY)
```

(CURRENT_DATE is equivalent to DATE 'now'.) This query retrieves all the current employees, that is, all those employee tuples that overlap DATE 'now' in valid-time. The semantics of the temporal variable |now| in the query requires that it be bound to the time when the query is evaluated. Thus for example, if the query is evaluated on July 9, then this query is equivalent to:

```
SELECT NAME, RANK
FROM FACULTY
WHERE DATE '07/09/1994' OVERLAPS VALID(FACULTY)
```

Note that the query might subsequently be evaluated at some other date, for example July 31, at which time the variable DATE 'now' would be bound to that time.

We do, however, provide a function that prevents *now* from being bound during evaluation of a query or update. The function is called nobind(). nobind() is a signal to the compiler to suspend generation of the "code" that binds a temporal value. nobind() can only appear in the target list of an insert or modify statement, and will generate a compile-time error if it appears elsewhere. We do not currently allow nobind() to appear in a select. To emphasize the difference between nobind() and its absence, consider the following three insertions.

```
INSERT INTO FACULTY VALUES (Jane, Assistant, PERIOD(DATE 'June 1', NOBIND(DATE 'now')))

INSERT INTO FACULTY VALUES (Jane, Assistant, PERIOD(DATE '06/01/1994', DATE 'now'))

INSERT INTO FACULTY VALUES (Jane, Assistant, PERIOD(DATE 'now', NOBIND(DATE 'now')))
```

Assume that all three updates were performed on June 13. The first update will store the tuple shown in Figure 1; the second update will store the tuple shown in Figure 4; and the third, Figure 5. In general, nobind() supports the insertion of now-relative datetimes, intervals, and periods into the database; without nobind() these temporal values would be bound during execution of a query or update.

FACULTY5		
NAME	*RANK*	*VALID-TIME*
Jane	Assistant	'[06/13/94 - *now*]'

Figure 5: Executing the insert with the period from `DATE 'now'` to `NOBIND(DATE 'now')`

FACULTY		
NAME	*RANK*	*TRANS-TIME*
Jane	Assistant	'[06/01/94 - now]'

Figure 6: Jane's employment tuple in a transaction-time relation

4 *Now* in Transaction Time

The transaction time concept that heretofore has been labeled with "now" is somewhat simpler than the valid time concept of *now*. The problem with *now* in transaction time is that the concept is misleadingly called "now."

Transaction time denotes the time period between a fact being stored in the database and the fact being (logically) deleted from the database [14]. It is an orthogonal concept to valid time, in that it concerns the history of the database, as opposed to the history of the enterprise being modeled.

Transaction-time timestamps are supplied automatically by the DBMS during updates (valid-time timestamps are generally supplied by the user). Specifically, insertions initialize the starting transaction time to the "current time" and the terminating transaction time to *now*. (There is an additional requirement that the transaction time be consistent with the transaction serialization order.) Updates change the terminating time of *now* to the value of the current transaction time. Hence, in transaction-time relations, deletion is logical. The information is not physically removed from the relation, rather it is tagged as no longer current by having a terminating time different from *now*. Physical deletion never occurs in a transaction-time relation.

As an example, consider the transaction-time relation shown in Figure 6. The distinct semantics of transaction time yields a different interpretation of this relation as compared with the one shown in Figure 1. The start of the transaction time period indicates that this tuple was stored in the database on 06/01/94, e.g., the database first became aware that Jane was a faculty member on that date. The period ends at *now*, indicating that we still believe that Jane is an Assistant professor at State University. When we learn on 07/10/94 that Jane left State University, we will logically delete this tuple by changing the period to `'[06/01/94 - 07/10/94]'`.

4.1 The Label "Now" in Transaction Time

In transaction time, a tuple timestamped with a terminating transaction time of *now* means that this tuple has not yet been logically deleted [23]. But the label "now" actually obscures this meaning. Strictly speaking, it implies that every current tuple was deleted by the current transaction! In Figure 6, if the current time is 07/09/94, then a strict interpretation of a terminating time of "now" suggests that the

FACULTY		
NAME	*RANK*	*TRANS-TIME*
Jane	Assistant	'[06/01/94 - forever]'

Figure 7: Using forever in a transaction-time relation

FACULTY		
NAME	*RANK*	*TRANS-TIME*
Jane	Assistant	'[06/01/94 - until changed]'

Figure 8: Using until changed in a transaction-time relation

terminating time is 07/09/94 (we used exactly the same interpretation for "now" in valid time). This is not what was intended.

4.2 The Label "Forever" in Transaction Time

As with valid time, some data models address this problem by using "forever" instead of "now," as shown in Figure 7 [15, 3, 4, 20]. And as before, we immediately encounter other difficulties. The strict interpretation of this tuple is that a transaction executing a (very) long time in the future will logically delete this tuple from the relation. In the meantime, it will remain in the database. If, on 07/10/94, it becomes known that Jane has left State University, then we logically delete this tuple by changing the terminating time to 07/10/94. Such a change is inconsistent with the previous terminating time, thus implying that the label "forever" is not an adequate solution. In this one sense, "now" is somewhat more appropriate.

4.3 The Label "Until Changed" in Transaction Time

A more precise label than "now" or "forever" for the transaction-time concept of "not yet logically deleted" is "until changed." The most recent transaction for a fact is considered the current state of that fact, *until changed* by some later transaction. Querying the current state, i.e., in a rollback operation, considers all tuples with a terminating time of *until changed*, and no other tuples. We advocate using the label "until changed" instead of the label "now" in transaction time to make clear the special, transaction-time specific meaning of *now*, and to ensure that updates are consistent with, and in fact a refinement of, currently stored information. For our running transaction-time example, this would appear as shown in Figure 8.

"Until changed" is a distinguished transaction-time literal, appearing in a datetime or period constant. It has no counterpart in valid time (using "until changed" instead of "now" avoids potential confusion with "now" in valid time, although some authors use "until changed" in valid time [21, 22]). Also, it can only be used as the terminating transaction time; it is nonsensical to use it as the starting time.

We emphasize that *until changed* is not a different variable than *now*, merely a different label for

the same variable. The label expresses the unique, transaction time semantics for the variable *now*.

5 Summary

In temporal databases *now* is a commonly used value. In this document we developed the concept of a now-relative datetime, outlined timestamp formats to store now-relative datetimes, and considered the impact of now-relative datetimes on query processing.

We propose that support for *now* be added to TSQL2. This support requires few changes to the data model, *now* will be handled by simply replacing it with the reference time during query or update evaluation. We allow an interval "offset" to be coupled with *now* resulting in a now-relative datetime which is indispensable to modeling some kinds of temporal information. We also advocate use of the unary function `NOBIND()` to distinguish the mention of *now* from its use in a query. We further noted a difference in the transaction-time and valid-time uses of *now*. To highlight this difference we recommend using "until changed" as the transaction-time label for *now* and "now" as the valid-time label. The naming convention is enforced by each calendar during input and output of temporal constants. Finally, we anticipate that timestamp operation efficiency will remain high, even for these complex datetimes.

In closing, we note that adding *now* to TSQL2 requires no schema level or syntax changes to the language and minimal changes to the temporal algebra. Some changes must be made to the timestamp representation and operations, but these changes were planned for in the initial timestamp design.

Acknowledgements

This work was supported in part by NSF grants ISI-8902707 and ISI-9302244, IBM contract #1124 and the AT&T Foundation.

References

[1] G. Ariav, A. Beller, and H.L. Morgan. A temporal data model. Technical Report DS-WP 82-12-05, Decision Sciences Department, University of Pennsylvania, December 1984.

[2] M.A. Bassiouni and M.J. Llewellyn. A relational-calculus query language for historical databases. *Computer Languages*, 17(3):185–197, 1992.

[3] J. Ben-Zvi. *The Time Relational Model.* PhD thesis, University of California at Los Angeles, 1982.

[4] G. Bhargava and S. Gadia. Achieving zero information loss in a classical database environment. In *International Conference on Very Large Databases*, pages 217–224, Amsterdam, August 1989.

[5] C. J. Date and C. J. White. *A Guide to DB2*, volume 1, 3rd edition. Addison-Wesley, Reading, MA, September 1990.

[6] C. E. Dyreson and R. T. Snodgrass. Timestamp semantics and representation. *Information Systems*, 18(3):143–166, 1993.

[7] C.E. Dyreson and R.T. Snodgrass. The TSQL2 Timestamp Representation. Technical report, TSQL2 Design Committee, 1993.

[8] R. Elmasri, G. Wuu, and Y. Kim. The time index - an access structure for temporal data. In *International Conference on Very Large Databases*, Brisbane, Australia, August 1990.

[9] S. K. Gadia. A homogeneous relational model and query languages for temporal databases. *Transaction on Database Systems*, 13(4):418–448, December 1988.

[10] C.S. Jensen, J. Clifford, S.K. Gaida, A. Segev, and R.T. Snodgrass. A glossary of temporal database concepts. *ACM SIGMOD Record*, 21(3), September 1992.

[11] J. (ed.) Melton. *Solicitation of Comments: Database Language SQL2*. American National Standards Institute, Washington, DC, July 1990.

[12] S. B. Navathe and R. Ahmed. A temporal relational model and a query language. *Information Sciences*, 49:147–175, 1989.

[13] N. Sarda. Algebra and query language for a historical data model. *The Computer Journal*, 33(1):11–18, February 1990.

[14] R. Snodgrass and I. Ahn. A taxonomy of time in databases. In *SIGMOD*, pages 236–246, Austin, TX, May 1985.

[15] R. T. Snodgrass. The temporal query language tquel. *Transactions on Database Systems*, 12(2):247–298, June 1987.

[16] R. T. Snodgrass. *An Overview of TQuel*, chapter 6. Benjamin/Cummings, 1993.

[17] M. Soo, C.E. Dyreson, and R. Snodgrass. User-defined time in tsql2. Technical report, TSQL2 Design Committee, 1993.

[18] J. B. Sykes, editor. *The Concise Oxford Dictionary*. Oxford University Press, Oxford, England, 1964.

[19] A U Tansel. Modelling temporal data. *Information and Software Technology*, 32(8):514–520, October 1990.

[20] S. Thirumalai and S. Krishna. Data organization for temporal databases. Technical report, Raman Research Institute, India, Bangalore, India, 1988.

[21] G. Wiederhold, S. Jajodia, and W. Litwin. Dealing with granularity of time in temporal databases. In *Proc. 3rd Nordic Conf. on Advanced Information Systems Engineering*, Trondheim, Norway, May 1991.

[22] G. Wiederhold, S. Jajodia, and W. Litwin. Integrating temporal data in a heterogeneous environment. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases*. Press, 1993.

[23] C. Yau and G. S. W. Chat. Tempsql – a language interface to a temporal relational model. *Information Sc. & Tech.*, pages 44–60, October 1991.

A Modified Language Syntax

The organization of this section follows that of the SQL2 document. The syntax is listed under corresponding section numbers in the SQL2 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 SQL2 proposal, and that a copy of the proposal is available for reference.

A.1 Section 5.2 <token>

One reserved word was added.

<reserved word> ::=
- | `NOBIND`

A.2 Section 5.3 <literal>

No new syntax is introduced, but the allowable datetime, interval, and period literals is expanded to support indeterminate values.

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

<interval string> ::=
- | <now-relative interval string>

Additional syntax rules:

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

   Case:

   - 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 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`.
   - Let A be a valid <determinate interval string>, representing the interval B. Let C be a string consistent with the *sign_format* property, which can include references to the field *sign*. If the value of the *now_relative_datetime_format* property, with the *now* field replaced with the value of the property *now_string*, the *determinate_interval* field replaced with A, and the *sign* field replaced with C, is identical to the <datetime string>, then the value represented by the <datetime string> is the now-relative datetime $now + B$ or $now - B$, dependening on whether the *sign* field value is 0 or 1.
   - Let A be a valid <now-relative datetime string>, representing the datetime B. Let C be a valid <determinate datetime string>, representing the datetime D. Let E be a string

consistent with the *distribution_format* property, which can include references to the field *distribution_name*. If the value of the *indeterminate_now_relative_datetime_format* property, with the *now_relative_datetime* field replaced with B, the *determinate_datetime* field replaced with D, and the *distribution* field replaced with E, is identical to the <indeterminate now-relative datetime string>, then the value represented by <indeterminate now-relative datetime string> is the indeterminate now-relative datetime with lower support B, upper support D, and distribution as named in E.

- Let A be a valid <indeterminate interval string>, representing the interval B, with lower support C, upper support D, and distribution E. Let F be a string consistent with the *sign_format* property, which can include references to the field *sign*. If the value of the *now_relative_with_indeterminate_interval_datetime_format* property, with the *now* field replaced with the value of the property *now_string*, the *indeterminate_interval* field replaced with A, and the *sign* field replaced with F, is identical to the <now-relative with indeterminate datetime string>, then the value represented by the <now-relative with indeterminate datetime string> is the indeterminate datetime with lower support *now* + C or *now* - C depending on whether the *sign* field value is 0 or 1, upper support D, and distribution E.

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

   Case:

   - 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.

A.3 Section 6.14 <datetime value function>

The nobind function is added.

<datetime value function> ::=

- | `NOBIND` <left paren> <datetime literal> <right paren>
- | `NOBIND` <left paren> <column reference> <right paren>

Additional general rules:

1. 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.

B Section 6.?? <interval value function>

The nobind function is added.

<interval value function> ::=

- | `NOBIND` <left paren> <interval literal> <right paren>
- | `NOBIND` <left paren> <column reference> <right paren>

Additional general rules:

1. 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.

C Section 6.?? <period value function>

The nobind function is added.

<period value function> ::=

- | `NOBIND` <left paren> <period literal> <right paren>
- | `NOBIND` <left paren> <column reference> <right paren>

Additional general rules:

1. 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.

C.1 Section 13.7 <delete statement: searched>

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.

C.2 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'))`.

C.3 Section 13.10 <update statement: searched>

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'))`.