18

Now

James Clifford, Curtis E. Dyreson, Richard T. Snodgrass, Tomás Isakowitz, and Christian S. Jensen

1 Introduction

Now is an English noun meaning "at the present time" [16]. *Now* is also a distinguished timestamp value in many temporal data model proposals. In this chapter, we give precise, but informal, semantics for this familiar term and discuss 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 in [15, Chapter 8] and [15, Chapter 5]. 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, 7, 8, 10, 11, 17, 19]. 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

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

Figure 1: Jane's employment tuple

the university's employment history (the FACULTY1 table). 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 chapter 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" [20, 21], but the semantics is the same.

Other data models use *forever* or ∞ as the terminating period datetime, as shown in Figure 2 [3, 13, 14][18]. Forever is the largest representable timestamp value, that is, the one furthest into the 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, 9]; in TSQL2's more liberal design, 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 [10]). 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.

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

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

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 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 TSQL2.

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

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

Figure 3: Using a now-relative datetime

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.,

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

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

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

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). TSQL2 does not 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 $-\infty$).

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 UPDATE 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.

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

Figure 4: Executing the insert with `date 'now'`

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')`

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 [12]. 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 tables, deletion is logical. The information is not physically removed from the table, rather it is tagged as no longer current by having a terminating time different from *now*. Physical deletion never occurs in a transaction-time table.

As an example, consider the transaction-time table shown in Figure 6. The distinct semantics of transaction time yields a different interpretation of this table 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]'`.

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

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

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

Figure 7: Using forever in a transaction-time table

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 [19]. 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 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 [3, 4, 13, 18]. 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 table. 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

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

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

"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 [20, 21]). 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 chapter 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.

TSQL2's support for *now* 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.

References

[1] Ariav, G., A. Beller and H. L. Morgan. “A Temporal Data Model.” Technical Report DS-WP 82-12-05. Decision Sciences Department, University of Pennsylvania. Dec. 1984.

[2] Bassiouni, M. A. and M. J. Llewellyn. “A Relational-Calculus Query Language For Historical Databases.” *Computer Languages*, 17, No. 3 (1992), pp. 185–197.

[3] Ben-Zvi, J. “The Time Relational Model.” PhD. Dissertation. Computer Science Department, UCLA, 1982.

[4] Bhargava, G. and S. Gadia. “Achieving Zero Information Loss in a Classical Database Environment,” in *Proceedings of the Conference on Very Large Databases*. Amsterdam: Aug. 1989, pp. 217–224.

[5] Date, C. J. and C. J. White. “A Guide to DB2.” Reading, MA: Addison-Wesley, 1990. Vol. 1, 3rd edition.

[6] Dyreson, C. E. and R. T. Snodgrass. “Timestamp Semantics and Representation.” *Information Systems*, 18, No. 3 (1993), pp. 143–166.

[7] Elmasri, R., G. Wuu and Y. Kim. “The Time Index - An Access Structure for Temporal Data,” in *Proceedings of the Conference on Very Large Databases*. Brisbane, Australia: Aug. 1990.

[8] Gadia, S. K. “A Homogeneous Relational Model and Query Languages for Temporal Databases.” *ACM Transactions on Database Systems*, 13, No. 4, Dec. 1988, pp. 418–448.

[9] Melton, J. and A. R. Simon. “Understanding the New SQL: A Complete Guide.” San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1993.

[10] Navathe, S. B. and R. Ahmed. “A Temporal Relational Model and a Query Language.” *Information Sciences*, 49 (1989), pp. 147–175.

[11] Sarda, N. L. “Algebra and Query Language for a Historical Data Model.” *The Computer Journal*, 33, No. 1, Feb. 1990, pp. 11–18.

[12] Snodgrass, R. T. and I. Ahn. “A Taxonomy of Time in Databases,” in *Proceedings of ACM SIGMOD International Conference on Management of Data*. Ed. S. Navathe. Association for Computing Machinery. Austin, TX: May 1985, pp. 236–246.

[13] Snodgrass, R. T. “The Temporal Query Language TQuel.” *ACM Transactions on Database Systems*, 12, No. 2, June 1987, pp. 247–298.

[14] Snodgrass, R. T. “An Overview of TQuel,” in Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, 1993. Chap. 6. pp. 141–182.

[15] Snodgrass, R. T. (editor) “The TSQL2 Temporal Query Language.” Kluwer Academic Publishers, 1995.

[16] Sykes, J. B. “The Concise Oxford Dictionary.” Oxford, England: Oxford University Press, 1964.

[17] Tansel, A. U. “Modelling temporal data.” *Information and Software Technology*, 32, No. 8, Oct. 1990, pp. 514–520.

[18] Thirumalai, S. and S. Krishna. “Data Organization for Temporal Databases.” Technical Report. Raman Research Institute, India. 1988.

[19] Yau, C. and G. S. W. Chat. “TempSQL – A Language Interface to a Temporal Relational Model.” *Information Sc. & Tech.*, , Oct. 1991, pp. 44–60.

[20] Wiederhold, G., 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.

[21] Wiederhold, G., S. Jajodia and W. Litwin. “Integrating Temporal Data in a Heterogeneous Environment,” in Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, 1993. Chap. 22. pp. 563–579.