# 17

# Cursors

**Christian S. Jensen, Richard T. Snodgrass, and T. Y. Cliff Leung**

The cursor facility of SQL2, to be useful in TSQL2, must be revised. Essentially, revision is needed because tuples of TSQL2 have associated valid-time elements, i.e., finite unions of periods.

The data model of TSQL2 was designed as a conceptually minimal valid-time extension of the standard relational model. Facts in a valid-time table have associated the times when they are assumed to be true in the modeled reality. Tuples encode facts (atomic or composite), and the adopted data model associates valid times with facts at the level of tuples. In order to not replicate facts, i.e., have several value-equivalent tuples with different associated time periods, tuples are associated with valid-time elements, finite unions of periods.

Storage and display of valid-time tables is not restrained by the above choice of data model. Indeed, several formats may coexist for both storage and display. As will be described, we have here adopted a very simple display format that is well-suited for accessing valid-time tables via cursors. Other display formats may be added later.

The association of tuples with valid-time elements necessitates modifications of the SQL2 cursor mechanism before it can be adopted for TSQL2. This chapter examines such modifications. When the cursor mechanism was designed, it was attempted to make only minimal modifications of the SQL2 cursors.

To illustrate the use of cursors in TSQL2, consider an example. First, a valid-time table schema and (empty) instance is defined which records the names of employees, their departments, and salaries.

```
CREATE TABLE Employee (Name CHAR(10),Dept CHAR(10),
   Salary INTEGER)
VALID AS PERIOD
```

Next, a tuple is inserted which records Ben's salary and indicates that he was in the Toy department in January, March, and May of 1993. A subsequent insertion records that he was in the Book department in February and April of 1993. His salary remains 30 throughout these periods. Conceptually, the table shown next results.

| Name | Department | Salary | V |
|---|---|---|---|
| Ben | Toy | 30 | [1 Jan 1993, 31 Jan 1993] ∪ [1 Mar 1993, 31 Mar 1993] ∪ [1 May 1993, 31 May 1993] |
| Ben | Book | 30 | [1 Feb 1993, 28 Feb 1993] ∪ [1 Apr 1993, 30 Apr 1993] |

We use this example for describing the design of the cursor mechanism of TSQL2.

For accessing valid-time tables, we will use the display format described next. The format resembles that of the table above. Specifically, the only addition is that timestamps are not finite unions of periods, but are instead lists of maximal periods. Further, each period is represented by pairs of delimiting datetimes. Each pair is ordered with the smallest value first. When a valid-time table is transformed as described, a cursor may access the full valid-time table by simply iterating (using `OPEN`, `FETCH`, and `CLOSE`) through the tuples. This is a very minimal extension.

For the example above, this table is created.

| Name | Department | Salary | |
|---|---|---|---|
| Ben | Toy | 30 | [1 Jan 1993, 31 Jan 1993] |
| Ben | Toy | 30 | [1 Mar 1993, 31 Mar 1993] |
| Ben | Toy | 30 | [1 May 1993, 31 May 1993] |
| Ben | Book | 30 | [1 Feb 1993, 28 Feb 1993] |
| Ben | Book | 30 | [1 Apr 1993, 30 Apr 1993] |

Next, it is convenient to map the period-valued attribute to two datetime-valued attributes. Thus, in the display format adopted for cursor manipulation, two unnamed attributes are assumed which contain the start and end valid times of the corresponding maximal periods. Thus, the table above would be mapped to the following.

| Name | Department | Salary | | |
|---|---|---|---|---|
| Ben | Toy | 30 | 1 Jan 1993 | 31 Jan 1993 |
| Ben | Toy | 30 | 1 Mar 1993 | 31 Mar 1993 |
| Ben | Toy | 30 | 1 May 1993 | 31 May 1993 |
| Ben | Book | 30 | 1 Feb 1993 | 28 Feb 1993 |
| Ben | Book | 30 | 1 Apr 1993 | 30 Apr 1993 |

To explore the mapping of a variable-length tuple to a fixed-length record format for retrieval purposes, recall first the (simplified) template for declaring a cursor.

We next explore how the contents of a valid-time table is accessed using cursors. To do so, recall the (simplified) template for declaring a cursor.

```
DECLARE cursor CURSOR FOR query-exp
```

For example, a cursor may be declared for employees named Ben.

```
DECLARE EmpCursor CURSOR
FOR SELECT *
   FROM Employee
   WHERE Employee.Name = ’Ben’
```

The syntax for cursor declaration is not affected. The following excerpts exemplify how the cursor may be used for retrieval.

```
FETCH NEXT FROM EmpCursor
INTO :nameVar, :deptVar, :salaryVar
```

This statement retrieves the explicit attribute values of the next tuple. Using this type of `FETCH`, it is possible to iterate through a table instance. The next excerpt indicates how the lists of pairs of timestamps associated with the tuples are retrieved.

```
FETCH NEXT FROM EmpCursor
INTO VALID :timeStartVar, :timeEndVar
```

This statement retrieves the next pair of timestamps of the current tuple. In a programming language, the second type of `FETCH` will typically be nested into a statement of the first type, forming a nested loop. The first type of `FETCH` is identical to the conventional `FETCH`, and status information is passed from SQL to the programming language via the `SQLCODE` parameter. Because the valid times associated with a tuple may be thought of as a binary table, the `SQLCODE` parameter is used for timestamps the same way it is used for tuples. The previous two statements may be combined.

```
FETCH NEXT FROM EmpCursor
INTO :nameVar, :deptVar, :salaryVar
VALID INTO :timeStartVar, :timeEndVar
```

This statement retrieves the explicit attributes of the next tuple as well as the bounding datetimes of the first valid-time period associated with the explicit-attribute values. This type of `FETCH` is especially useful when a cursor is declared to range over a table instance where all the valid-time elements associated with tuples are guaranteed to consist of single periods.

We have seen that a valid-time element of a tuple is passed using cursors to an application program by passing each maximal period in turn. By default, a period is passed by passing the delimiting datetimes. This default has been adopted because application programs are expected to find this format most useful. However, a period may also be passed to a period-valued programming language variable, as illustrated next.

```
FETCH NEXT FROM EmpCursor
INTO VALID PERIOD :periodVar
```

When SQL statements are embedded in a programming language, they are prefixed by `EXEC SQL` in all languages. This prefix was omitted above because different programming languages indicate the termination of the SQL statement differently (using, e.g., `END-EXEC` or "`;`"). Variables defined in the surrounding program (written in, e.g., PL/1) are prefixed by "`:`".

*Cursor stability*, a special, weak level of consistency, may also be applied in the context of TSQL2 cursors. Employing cursor stability in a transaction that is iterating through entire tables may enhance system performance. In snapshot databases, cursor stability ensures that the tuple that is currently being processed is protected from writes (i.e., it is locked in shared mode) and that tuples modified by the transaction are locked in exclusive mode until the transaction is terminated.

We have added a nested cursor mechanism to TSQL2. Thus, the notion of a current tuple in the context of cursor stability must be redefined. Specifically, we define the current tuple to be the tuple indicated by the outer cursor. Thus, the current tuple is a complete tuple. It has associated a complete valid-time element, not just the current period as indicated by the inner, timestamp-level cursor. Together, the outer and inner cursors for a query expression generally point to a partial tuple, namely an explicit-attribute component and a single valid-time period.

In TSQL2, cursor stability ensures that the part of the current tuple pointed to by the nested cursor is protected from writes (i.e., it is locked in shared mode). Further, if any part of a tuple is modified by the transaction, the complete tuple is locked in exclusive mode until the transaction is terminated.

As indicated above, cursors also play a role during deletion and update. Specifically, cursors are utilized during position delete and update statements. For example, the template for positioned delete is given as follows.

```
DELETE FROM <table name>
WHERE CURRENT OF <cursor name>
```

The delete and update statements are related to cursors because they manipulate the current tuple as indicated by the cursor (i.e., by `WHERE CURRENT OF` <cursor name>). In these statements, as above, the current tuple is always a complete tuple, namely the tuple indicated by the outer cursor (together with the complete associated valid-time element) Consequently, positioned deletions as well as updates apply to complete tuples.