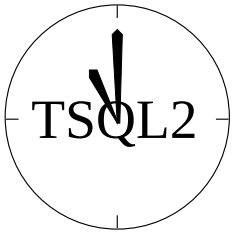

Valid-time Projection in TSQL2

September 16, 1994

A TSQL2 Commentary

The TSQL2 Language Design Committee

Title	Valid-time Projection in TSQL2
Primary Author(s)	Suchen Hsu, Christian S. Jensen and Richard Snodgrass
Publication History	May 1992. TempIS Document No. 30. 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

Contents

List of Tables

Abstract

Temporal databases have now been studied for more than a decade, and numerous temporal query languages have been proposed. *Valid-time projection*, which defines the timestamps of the tuples in query results, is an important ingredient of a temporal query language. Often, valid-time projections is closely tied to another important component, namely *valid-time selection*, which allows the user to retrieve tuples based on their underlying valid-times. We have previously surveyed valid-time selection and projection in nine temporal query languages, primarily SQL and Quel extensions. Based on that survey, this document proposes a specific design of the valid-time projection component of the consensus temporal query language TSQL2 that is currently being designed.

1 Introduction

We have previously examined the valid-time selection and projection components of nine different temporal query languages. The examination established a foundation for designing the valid-time selection and projection components of the consensus Temporal SQL2 that is currently being designed. We have already proposed a specific design for valid-time selection in TSQL2. This document proposes a specific design for the valid-time projection component. It is attempted to base the proposal on the experiences accumulated in previous proposals, and an effort is made to explicitly address important design decisions.

The document is structured as follows. Initially, we describe six general criteria for what is a good language design. The criteria provide guidelines. However, they may conflict at times, and a single general criterion may be applied in several ways in a concrete design. Then, the proposal for valid-time projection is introduced. Following an overview, the motivation for adding a separate clause for valid-time projections is discussed, and the choice of defaults is considered in some detail. The document is ended with a summary.

2 Design Criteria

As a guide for making appropriated design decisions and as a means of evaluating the proposal, we present some language design criteria. These criteria are *expressive power, consistency, clarity, minimality, orthogonality,* and *independence.* Initially, each criterion is described. Then the interactions among the criteria are exemplified.

Expressive Power This criterion indicates that the language must exhibit a functionality that makes it suitable for its intended applications and does not impose undesirable restrictions on the queries that may be formulated.

This does not mean that providing a lot of operators and functions is necessarily better than a more restricted set. For example, this criterion has influenced us when we decided to include a separate clause for valid-time projection.

Consistency For the task at hand, this criterion has at least four implications. First, the design must be consistent with the syntax for user-defined time support in TSQL2. For example, it should use the formats for temporal constants adopted there [Soo & Snodgrass 1992]. Second, the design should

be upward compatible with SQL2. This indicates that defaults should be chosen carefully. Third, the proposal should be consistent with the designs of other aspects of TSQL2. Fourth, the syntax should be internally consistent.

Clarity The syntax should clearly reflect the semantics of the language. This aids in formulating and understanding queries. Applications of the principle include the meaningful naming of operators, a proper choice of clauses (to obtain well-structured queries), and a consistent naming style.

Minimality The principle of minimality indicates that as few as possible new reserved words and clauses should be introduced and added to those already present in SQL2. It also indicates that new operators should not be included if they duplicate the functionality already provided by existing operators. This is intended to ensure that users will not be unnecessarily burdened by a large redundant set of options.

Orthogonality It should be possible to freely combine query language constructs that are semantically independent. For example, we consider valid-time selection and projection semantically independent and have thus separated these. On the other hand, we do not consider temporal and non-temporal selection independent.

The Zero–One-Infinity principle may be seen as a more specific design criterion. This criterion states that the only reasonable numbers in a design are zero, one and infinity and that other numbers are unintuitive to users. For example, restricting the number of tuple variables that may be declared in a query to another number (e.g., 15) appears to have no logical explanation and is difficult to remember.

Independence Obeying this criterion ensures that each function is accomplished by only one way. Designing functions to be independent and non-overlapping, orthogonality, minimality, and consistency may be achieved.

Although we would like the design to satisfy all of the criteria, this is not possible because the criteria themselves are conflicting.

3 Valid-time Projection in TSQL2

The appendix (Section A) describes the syntax modifications to the SQL2 language necessary to include valid-time projection. Valid-time projection is specified in the optional <valid clause> (Section A.2). This clause consists of either `VALID` or `VALID INTERSECT` followed by a temporal expression.

3.1 Overview

Unlike some other proposed temporal query languages, we do not distinguish syntactically between instant and period projection. There are two reasons for this. First, in standard SQL, no clauses have combinations of reserved words. Second, the compiler is capable of determining the correct type of a temporal expression and then define the correct timestamp type for the resulting relation.

However, we do support two different options in the valid clause: `VALID` and `VALID INTERSECT`. The reserved word `VALID` indicates a general valid-time projection, and all valid-time expressions returning periods or elements are allowed in the clause.

The alternative `VALID INTERSECT` indicates a restriction on the projection, namely that the resulting timestamp is the intersection of the time of the specified temporal expression and the timestamps of the relations (actually, correlation names) appearing in the FROM clause. If the given timestamps and temporal expression do not intersect, the resulting timestamp is empty, and hence that tuple doesn't participate further in the query. For example, the following two statements are equivalent.

`VALID INTERSECT` < *temporal expression* >
`VALID INTERSECT(`<*temporal expression*>`, INTERSECT(` <*argument relations*>`))`

From this, it is clear that `VALID INTERSECT` is subsumed by `VALID`. The reason for still having the alternative `VALID INTERSECT` is that it will be used very frequently in queries that should return tuples with valid times that do not extend beyond the valid times of the argument tuples. Put differently, using `VALID INTERSECT` ensures that queries do not return "manufactured" information that was not present in the database. Alternatively, using `VALID INTERSECT` restricts the possibilities for assigning timestamps to resulting tuples.

The `VALID` form is useful when new information, e.g., to be entered into the database, is computed from existing information. The following is a sample use of `VALID`.

Q3. Create a new department, Newtoy, from the original Toy department so that all of the employees currently in the Toy department will work in Newtoy one month from now.

```
INSERT
INTO Newdept(Name, Dept)
SELECT Name, 'Newtoy'
VALID PERIOD(CURRENT_DATE + INTERVAL '1' MONTH, DATE 'forever')
FROM Employee
WHERE Dept = 'Toy' AND OVERLAP(Employee, CURRENT_DATE)
```

In this example, we use the `VALID` form to specify the valid time for the new relation, `Newdept`. The `INTERVAL '1' MONTH` is an interval literal and indicates a duration of one month [Soo & Snodgrass 1992].

The timestamp in valid-time projection is specified as a valid-time expression which could be either period or element time.

The syntax and semantics of the valid-time expression in the valid clause is the same as the valid-time expressions specified in the `WHERE` clause. Thus, the period and element functions, as well as arithmetic operators, can all be used in valid-time projections.

The default value of the valid clause is the intersection of the timestamps of the argument relations, i.e., omitting `VALID` (`VALID INTERSECT`) is equivalent to "`VALID NTERSECT(`$relation_1$`,` $relation_2$`)`." If at least one of the argument relations is a snapshot relation, the default is also a snapshot relation. Table 1 shows the default value of the following (generic) query.

```
SELECT R1.A, R2.B
FROM R1, R2
WHERE ...
```

R1	R2	default value
valid time relation	valid time relation	INTERSECT(R1, R2)
valid time relation	snapshot relation	snapshot
snapshot relation	snapshot relation	snapshot

Table 1: Default Values for the VALID Clause

If the user wants a snapshot relation as a result of a query where the argument relations are valid-time relations, the reserved word SNAPSHOT is specified after SELECT. This use of SNAPSHOT is similar to the current use of DISTINCT. For example, the following query lists only the employees who have ever worked in the Toy department, with no timestamps in the resulting relation.

```
SELECT SNAPSHOT NAME
FROM EMP
WHERE DEPT = 'Toy'
```

In this section, we have introduced the valid clause; in the next section, we will motivate in more detail why the clause was included into the language.

3.2 Why a New Clause?

Originally, we wanted to add valid-time projection to the SELECT clause because valid-time projection is a kind of projection. This design would be consistent with the original SQL syntax.

However, it introduces a conflict with the earlier design decision of using the tuple-variable name for timestamp referencing. Adding valid-time projection to the SELECT clause would mean that some attributes (i.e., the explicit, non-timestamp attributes) would be referenced via their names while the valid timestamp is referenced via the tuple-variable name.

Including the valid-time projection in the SELECT clause also implies that the valid time of a tuple is simply a regular attribute, not the underlying valid time of the entire tuple. On the other hand, the chosen design is consistent with the over-all special treatment of valid time in the query language.

Another disadvantage is that we need to add one reserved word for naming valid time in situations where a new relation is created. For example, **Q3** could be written as follows, where the reserved word VALIDTIMESTAMP is employed to indicate the implicit valid time attribute of the resulting relation.

```
INSERT
INTO NEWDEPT(NAME, DEPT, VALIDTIMESTAMP)
SELECT NAME, 'NEWTOY', CURRENT_DATE + INTERVAL '1' MONTH
FROM EMPLOYEE
WHERE DEPT = 'Toy' AND OVERLAP(EMPLOYEE, CURRENT_DATE)
```

In summary, the major advantage of a new clause is clarity—where to define or read a valid-time projection is indicated clearly. In addition, having a new clause means that the `SELECT` clause is not complicated by a possibly lenghty valid-time projection. The presence of a new clause also emphasizes that the the valid time is not just another attribute, but is about the entire tuple.

One disadvantage is that a reserved word has been added. Another is that having a new clause for valid time may be claimed to be inconsistent with not having a new clause for valid-time selection.

Comparing the advantages and disadvantages of the two designs, we choose to include a separate clause for valid-time projection.

3.3 Discussion of Default Values

There are two obvious choices of default values for valid-time projection (the valid clause using the reserved word `VALID`). The first is the intersection of timestamps of the argument relations. The second is a snapshot relation, i.e., including no timestamp in the result relation. For example, the semantics of the following query is different under the two default values.

```
SELECT R1.A, R2.B
FROM R1, R2
WHERE R1.A = R2.C
```

If the default is intersection, the result is a valid-time relation, and the timestamp of each tuple is the intersection of timestamps of tuples from `R1` and `R2`. However, if the default is a snapshot, the result is a snapshot relation with two attributes, `A` and `B`.

Table 2 is an overview of the differences between the two possible defaults. The first column indicates the types of two argument relations in a query. These can be either valid-time relations (VTR) or snapshot relations (SR). The last column indicates the type of timestamp accorded result tuples, with "snapshot" indicating not timestamps, "others" indicating some unspecified timestamp, and "R1 ∩ R2" indicating the intersection of the timestamps of the argument tuples. The two middle columns show what must be written in the valid clause to achieve certain kinds of timestamps given certain kinds of argument relations. The first column assumes that the default is "snapshot" and the second assumes that "intersection" is the default. In these two columns, "default" means that no clause is present, and "temporal expression" denotes an arbitrary valid-time expression.

Type of R1 and R2	Expression in the valid clause		Timestamp of the result
	Default is "snapshot"	Default is "R1 ∩ R2"	
R1, R2: VTR	intersect(R1,R2)	default	R1 ∩ R2
R1, R2: VTR	default	?	snapshot
R1, R2: VTR	temporal expression	temporal expression	others
R1, R2: SR	default	default	snapshot
R1:VTR, R2: SR	default	default	snapshot
R1:VTR, R2: SR	temporal expression	temporal expression	others

Table 2: Possible Defaults for Expressions in the Valid Clause

The table shows that there is not much difference between the two defaults. For the queries covered by rows three to six, either default works the same way. There is a difference only in the first two rows

where the two argument relations are valid-time relations and where the desired result is a snapshot or a valid-time relation with intersection timestamps.

Because intersection of timestamps is believed to be used more often than producing snapshots, we choose the intersection of timestamps as the default value. The question mark in the second row indicates that some special mechanism is needed when a snapshot relation is to be produced from two valid-time relations.

Our solution is to add a new reserved word `SNAPSHOT` after `SELECT` to indicate that a snapshot relation is desired. The valid clause is simply left out. With the new reserved word, the syntax shows the resulting relation type explicitly. If we add the reserved word to the valid clause, the semantics are less clear. The reserved word is not a temporal expression and this is not consistent with the syntax of the valid clause. In SQL, the reserved word `DISTINCT` is also specified after `SELECT`. Thus, adding `SNAPSHOT` is consistent with SQL.

4 Summary

In a previous paper, we reviewed nine temporal query languages proposed in the past decade. These languages include five extensions to SQL, three extensions to QUEL, and a procedural language. Because their underlying data models were not the same, it is not easy to compare the features in each language. However, all of the features, functionalities, new clauses, and reserved words of these languages were examined carefully when this proposal for valid-time and projection in TSQL2 was prepared (a separate document describes a proposal for valid-time selection). I has been a goal to build on the insights gained from the designs of previous temporal query languages.

Initially, six criteria, expressive power, consistency, clarity, minimality, orthogonality, and independence, were defined in order to guide the design.

It was attempted to design simple but powerful language constructs with no unnecessary reserved words, clauses, and functions. Key features included a clean separation of valid-time selection and valid-time projection. The orthogonality of these constructs is reflected clearly in the design. A new clause for valid-time projection, with one reserved word, `VALID`, added. The former allows for assigning arbitrary timestamps to result tuples. The latter ensures that timestamp values of resulting tuples do not exceed the intersection of the timestamps of the argument tuples, i.e., it in not possible to manufacture information. We have thus allowed full flexibility (with `VALID`), but also have separated the "safe" queries (using `VALID INTERSECT`) from the potentially unsafe. Defaults have been chosen carefully. Most notably, a new reserved word `SNAPSHOT` is placed after `SELECT` to indicate that the result of a query should be a snapshot relation.

The proposal did not address the transfer of timestamp values to data structures of a programming language program execution via cursors. In SQL2, cursors can return values of explicit attributes; an extension is required to support the transfer of timestamp values.

5 Acknowledgements

We thank Curtis Dyreson, Nick Kline, and Michael Soo for discussions, suggestions, and comments. This work was supported in part by NSF grants ISI-8902707 and ISI-9302244, IBM contract #1124 and

the AT&T Foundation. Christian S. Jensen was also supported in part by the Danish Natural Science Research Council, grants no. 11–1089–1 SE and no. 11–0061–1 SE.

6 Bibliography

[Allen 1983] Allen, J.F. "Maintaining Knowledge about Temporal Intervals." *Communications of the Association of Computing Machinery*, 26, No. 11, Nov. 1983, pp. 832–843.

[Ariav 1986] Ariav, G. "A Temporally Oriented Data Model." *ACM Transactions on Database Systems*, 11, No. 4, Dec. 1986, pp. 499–527.

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

[Chamberlain et al. 1976] Chamberlain, D. D. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control." *IBM J.*, 20, No. 6 (1976), pp. 560–575.

[Gadia & Vaishnav 1985] Gadia, S.K. and J.H. Vaishnav. "A Query Language for a Homogeneous Temporal Database," in *Proceedings of the ACM Symposium on Principles of Database Systems*. Mar. 1985, pp. 51–56.

[Gadia 1988] 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.

[Gadia 1992] Gadia, Shashi K. "A Seamless generic extension of SQL for querying temporal data." preliminary. Computer Science Department, Iowa State University. Mar. 1992.

[Held et al. 1975] Held, G.D., M. Stonebraker and E. Wong. "INGRES–A Relational Data Base Management System," in *Proceedings of the AFIPS National Computer Conference*. Anaheim, CA: AFIPS Press, May 1975, pp. 409–416.

[Jones et al. 1979] Jones, S., P. Mason and R. Stamper. "LEGOL 2.0: A Relational Specification Language for Complex Rules." *Information Systems*, 4, No. 4, Nov. 1979, pp. 293–305.

[Martin et al. 1987] Martin, N.G., S.B. Navathe and R. Ahmed. "Dealing with Temporal Schema Anomalies in History Databases," in *Proceedings of the Conference on Very Large Databases*. Ed. P. Hammersley. Brighton, England: Sep. 1987, pp. 177–184.

[Navathe & Ahmed 1987] Navathe, S. B. and R. Ahmed. "TSQL-A Language Interface for History Databases," in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 113–128.

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

[Sarda 1990A] Sarda, N. "Extensions to SQL for Historical Databases." *IEEE Transactions on Knowledge and Data Engineering*, 2, No. 2, June 1990, pp. 220–230.

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

[Sarda 1990C] Sarda, N. "Time-rollback using Logs in Historical Databases." Technical Report. Indian Institute of Technology. June 1990.

[Sarda 1990D] Sarda, N. "Design of a Historical Database Management System," in *Proceedings of the CSI Indore Chapter Conference (invited paper).* Aug. 1990.

[Snodgrass & Ahn 1985] Snodgrass, R. 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.

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

[Soo & Snodgrass 1992] Soo, M. and R. Snodgrass. "Mixed Calendar Query Language Support for Temporal Constants." TempIS Technical Report 29. Computer Science Department, University of Arizona. October 30, 1991 1992.

[Tansel & Arkun 1986] Tansel, A.U. and M.E. Arkun. "HQuel, A Query Language for Historical Relational Databases," in *Proceedings of the Third International Workshop on Statistical and Scientific Databases.* July 1986.

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 standard, as well as with the user-defined time proposal, and that a copy of the standard and the proposal is available for reference.

A.1 Section 5.2 <token>

One reserved word is added.

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

A.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> }

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 value, the values from the other value-equivalent tuples are gathered into a temporal element.

A.3 Section 7.9 <query specification>

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

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

Additional general rules:

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

A.4 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>.