ANSI X3H2-96-013r1
ISO/IEC JTC1/SC21/WG3 DBL MCI-99

I S O
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION

April 9, 1996

**Subject:** SQL/Temporal

**Status:** Informational

**Title:** A Road Map of Additions to SQL/Temporal

**Source:** ANSI Expert's Contribution

**Author:** Richard T. Snodgrass

**Abstract:** This document outlines a "road map" of future change proposals to SQL/Temporal, drawn from the TSQL2 language specification. It also addresses ten common questions about temporal query languages.

# References


[1] Böhlen, M. H. *Valid Time Integrity Constraints.* Technical Report TR 94-30. Department of Computer Science, University of Arizona, Tucson, November, 1994.

[2] Böhlen, M. H., C. S. Jensen and R. T. Snodgrass. *Evaluating the Completeness of TSQL2.* In *Proceedings of the VLDB International Workshop on Temporal Databases.* Ed. J. Clifford and A. Tuzhilin. VLDB. Springer Verlag, September, 1995.

[3] Clifford, J. and A. Tuzhilin (editors). Proceedings of the VLDB International Workshop on Temporal Databases, Springer Verlag Workshops in Computing Series, Zürich, Switzerland, September, 1995.

[4] Jensen, C. S., J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes and S. Jajodia (editors). *A Glossary of Temporal Database Concepts. ACM SIGMOD Record 23*, No. 1, March, 1994, pp. 52–64.

[5] Melton, J. (ed.) *SQL/Temporal.* October, 1995. (ISO/IEC JTC 1/SC 21/WG 3 DBL-LHR-009.)

[6] Silberschatz, A., M. Stonebraker, and J. Ullman (editors). *Database Research: Achievements and Opportunities Into the 21st Century.* Report of an NSF Workshop on the Future of Database Systems Research, May, 1995.

[7] Snodgrass, R. T. and H. Kucera. *Rationale for Temporal Support in SQL3.* 1994. (ISO/IEC JTC1/SC21/WG3 DBL SOU-177, SQL/MM SOU-02.)

[8] Snodgrass, R. T. (editor), Ilsoo Ahn, Gad Ariav, Don Batory, James Clifford, Curtis E. Dyreson, Ramez Elmasri, Fabio Grandi, Christian S. Jensen, Wolfgang Käfer, Nick Kline, Krishna Kulkarni, T. Y. Cliff Leung, Nikos Lorentzos, John F. Roddick, Arie Segev, Michael D. Soo and Suryanarayana M. Sripada. The Temporal Query Language TSQL2. Kluwer Academic Pub., 1995, 674+xxiv pages.

[9] Snodgrass, R. T., M. H. Böhlen, C. S. Jensen and A. Steiner. *Adding Valid Time — Part A.* SQL/Temporal change proposal, ANSI X3H2-95-485, ISO/IEC JTC1/SC21/WG3 DBL LHR-096, December, 1995, 40 pages.

[10] Steiner, A. and M. H. Böhlen. The TimeDB Temporal Database Prototype, September, 1995. Available at `ftp://www.iesd.auc.dk/general/DBS/tdb/TimeCenter` or at `ftp://ftp.cs.arizona.edu/tsql/timecenter/TimeDB.tar.gz`.

[11] Tsotras, V. J. and A. Kumar. *Temporal Database Bibliography Update. ACM SIGMOD Record 25*, 1, March, 1996, 11 pages.


# 1 Introduction

Several members of the ISO committee have requested a "road map" of where SQL/Temporal is going, in terms of the next few change proposals. This document provides such a road map, and also addresses several common questions concerning temporal query languages.

# 2 Ten Questions

When application builders, database administrators, and members of standards committees alike first encounter a new concept, many questions naturally arise. Here we present, and answer, the ten most common questions concerning temporal support in SQL.

1. *Why temporal?*

   Most database applications store historical data. In fact, it is difficult to identify applications that *do not* require temporal support. Temporal requirements are more prevalent than requirements for spatial support, for textual support, or for multimedia support, all of which are being added to SQL3. The Gartner Group Inc. has found that on average one in every 50 lines of application code has a date reference. [7] and chapter 1 of [8] list a wide variety of applications that require temporal support.

**2.** *Is SQL (SQL-92, SQL3) adequate?*

That all of these applications have been written using SQL provides concrete evidence that temporal support in the query language is not mandatory. In fact, many applications storing historical data have been developed using SQL-89, which doesn't even include datetime or interval data types.

However, when considering adequacy, rather than necessity, the conclusion of the research community and of many users is that SQL does not provide adequate support for temporal applications. To illustrate this, the reader is invited to attempt to formulate the following straightforward, realistic queries in SQL3. In doing so, the limitations of SQL will be immediately apparent.

- An Employee table has three columns: Name, Manager and Dept. We then store historical information by adding a fourth column, When, of data type `PERIOD`. Manager is a foreign key for Employee.Name. This means that at each point in time, the character string value in the Manager column also occurs in the Name column (probably in a different row) at the same time. This cannot be expressed via SQL's foreign key constraint, which doesn't take time into account.
- Consider the query "List those employees who are not managers." This can easily be expressed in SQL, using `EXCEPT` or `NOT EXISTS`, on the original, three-column table. Things are just a little harder with the When column; a where predicate is required to extract the current employees. Now consider the query "List the history of those employees who are *or were* not managers." `EXCEPT` and `NOT EXISTS` won't work, because they don't consider time. This simple temporal query is a challenging one even to SQL experts.
- Consider the query "Give the number of employees in each department." Again, this is a simple query in SQL. Converting it into a temporal query "Give *the history of* the number of employees in each department" is extremely difficult without temporal support in the language.
- Consider the modification "Change the manager of the tools department for 1994 to Bob." This modification is difficult in SQL because only a portion of many validity periods needs be changed, with the information outside of 1994 retained.
- To know when rows are inserted and (logically) deleted, we add two more columns, InsertTime and DeleteTime, both of the data type `TIMESTAMP`. Of course, adding these two columns breaks the referential integrity constraint between Manager and Name (the manager must also be an employee). The referential integrity must be formulated as a complex assertion that takes the three time columns into account.
- We find out that the telephone bill for a department is unusually high, so we formulate a query, "How many employees are in each department," to get a start. This query is more difficult now that we have to deal with the time columns.
- It turns out that one of the departments shows an unreasonable number of employees. Is this inconsistency in the database widespread? How long has the database been incorrect? The query, "When did we think that departments are overly large?" is exceedingly hard to express in SQL.

Most users know only too well that while SQL is an extremely powerful language for writing queries on the current state, the language provides much less help when writing temporal queries, modifications, and constraints. What users don't realize is that appropriate language facilities have been developed that enable all of the above to be simple extensions of their nontemporal variants.

An intermediate SQL programmer can express the non-temporal analogues of the above seven statements in SQL in perhaps fifteen minutes. However, even SQL experts find these same queries challenging to do in several *hours* when time-varying data is taken into account. With the language extensions to be proposed, once again an intermediate SQL programmer would have no trouble expressing these in just a few minutes.

[7] and chapters 1, 3 and 4 of [8] provide further examples of queries that are difficult in SQL but easy to express using temporal constructs.

**3.** *Why TSQL2?*

Given that temporal constructs are needed, the focus turns to identifying which constructs make the most sense. This has been an active area of research for twenty years. In that time, over 1200 papers have been published [11]. Many temporal query languages have appeared in the literature. Every obvious approach, and many not so obvious, has been explored in depth.

In 1993, as a follow-on to the International Workshop on an Infrastructure for Temporal Databases, a group of eighteen temporal database experts initiated the design of a second-generation temporal query language as an extension of SQL. This work culminated in the design of TSQL2, whose language specification was published in October, 1994. Subsequently, a 700-page book detailing the design decisions of the language appeared in September, 1995 [8]. TSQL2 is by far the most comprehensive, well-documented, consensual temporal query language. It has become the de facto standard in the temporal database community. In papers presented at the recent VLDB Workshop on Temporal Databases, it was the only query language that was discussed by people other than its designers; it was the topic of 30% of the papers and panels at that workshop [3]. The distinguished authors of [6] summarized the situation as follows.

> "Long-term exploration for appropriate models of temporal data has now resulted in a number of different proposals for extending query languages to better support temporal data. Among these, the one which currently has the widest support is the TSQL2 proposal, which extends SQL-92." ([6], page 3)

TSQL2 emphasizes several core concepts. Central is the notion of *tables with temporal support*, i.e., having implicit timestamps. These timestamps are stored as additional columns, and, as we shall see, are accessible as such if so desired.

**4.** *Why not explicit timestamps?*

As we have seen above, a period column can certainly be used to store temporal information and to perform temporal queries and modifications. However, by adopting implicit timestamps, a number of significant advantages accrue. This is the reason 95% of the temporal models in the literature assume time is not in an explicit column.

- *Temporal upward compatibility*
  This notion has been defined formally [9]. Intuitively, when a table not storing temporal information, such as the three-column employee table, is rendered temporal, we would like existing (non-temporal) applications to work unchanged. While an SQL view can hide the fact that a When column has been added (using a where clause to select the current employee, department, and manager), this approach doesn't work for modifications, because such a view is not updatable. Hence, when explicit timestamps are added to a table to enable it to record time-varying data, existing applications are broken. Potentially hundreds of thousands of lines of code have to be carefully inspected to determine what changes to make. When timestamps are implicit, all existing non-temporal applications will continue to work, without change.
- *Sequenced queries*
  This notion is also defined formally [9]. Intuitively, a sequenced query is the temporal analogue of a query on the current state. The queries given above are sequenced queries. If timestamps are implicit, then the user is freed from being required to compute the resulting timestamps; instead, that task is the DBMS's responsibility. This considerably simplifies many queries. In particular, to convert a conventional query into a sequenced query, all one has to do is prepend the reserved word `VALID` to the conventional query. This greatly increases the ease of use of the query language. Any conventional query, however complex, utilizing whatever SQL3 constructs, can be so converted to a sequenced query, providing a time-varying result.

- *Sequenced modifications*
  Modifications that are to be applied over all time (e.g., correct someone's misspelled name), as well as modifications over a specified period of time (e.g., the modification above, applied to 1994) are very easy to express when the DBMS is responsible for modifying the timestamp. Such modifications become difficult when the user is responsible for computing the correct timestamp.
- *Integrity constraints*
  Similarly, integrity constraints that apply at each point in time (e.g., the referential integrity constraint above) or that apply at each point over a specified period of time are very easy to express with implicit timestamps [1]. The alternative is for the user to include the explicit column in the integrity constraint, which makes its specification very cumbersome. In particular, the available constructs to declare primary keys, uniqueness, and foreign keys and referential integrity do not apply to tables with explicit timestamp columns; such constraints need to be specified using complex assertions. An example is the foreign key constraint mentioned above on page 3.
- *Transaction time*
  The transaction time of a fact is the time between when it was inserted into the database and when it was logically removed from the database [4]. For some data, it is useful to retain past states, say for auditing purposes; in some situations, data retention is required by law. If the transaction time is present in a column, available for manipulation by the user, then it is impossible to ensure the correct semantics of transaction time. Users could easily corrupt the database. It is important to let the DBMS handle the semantics of transaction time.

  We note in passing that the benefits of sequenced queries also apply to transaction time. One can convert a conventional query to be *transaction sequenced* by prepending the reserved word `TRANSACTION` to the query. TSQL2 treats the two orthogonal concepts of valid time and transaction time symmetrically.

Most applications are temporal applications, and will be simplified considerably when the query language has adequate temporal support.

**5.** *What if a user prefers the explicit column approach?*

It is perfectly fine to use explicit period columns in TSQL2. TSQL2 does not replace any functionality in SQL; it only adds functionality that application developers can choose to use or to ignore. In particular, the following is possible.

- Conventional tables with a period column that specifies when the row is valid can be converted, via a view or within a query, to a table with implicit timestamps, to utilize the increased functionality (such as sequenced queries) available to tables with temporal support.
- Tables with valid-time support can be converted, via a view or within a query, to a conventional table with an additional period column, if the user prefers to manipulate the data in that fashion.

These conversions are termed *nonsequenced queries*, and are discussed briefly below.

We emphasize that this is not an either/or situation. Users can specify tables with temporal support if they wish the DBMS to take care of computing the appropriate timestamps, or they can specify conventional tables if they prefer to compute the timestamps themselves. This decision can be made at any time, including for an individual correlation name in a from clause within a particular query.

**6.** *Wouldn't a different language design also work?*

There is a plethora of language designs, with over 40 in the published literature. The TSQL2 language design committee included as members the designers of about a third of the extant temporal query languages, including ChronoBase, DM/T, HRDM, IXSQL, OQL, SQL+T, TEER, TMQL, TOOA, TOOSQL, TOSQL, and TQuel. The members of this committee have been studying and contributing to the field of temporal databases for a total of 165 years. They were aware of what works, and more importantly, what doesn't work. Many other designs have been explored. Seemingly attractive alternatives have turned out to have had undesirable properties. Temporal query language design is amazingly subtle (otherwise, there wouldn't be so many temporal query languages). That is why it is important to be cognizant of the substantial prior research, to avoid reinventing the wheel, or more likely, reinventing the triangular wheel.

[8] contains an extensive discussion of the design decisions of TSQL2, including an in-depth comparison with the many other temporal query languages that have been defined. This design has carefully avoided pitfalls exhibited by earlier languages, many of which are documented in the literature.

At the same time, it is important that the change proposals be responsive to the concerns raised by the standards committees and others. However, alterations to TSQL2 must be made very carefully, with a careful analysis of the ramifications of these changes.

As an example, the ANSI committee raised the concern at the December, 1994 meeting that in TSQL2 the reserved word `SNAPSHOT` was required to get the same behavior with tables with temporal support that was exhibited by tables without such support. Several members of the research community investigated further, and determined that that was indeed a problem with TSQL2 [2]. The change proposals, e.g., [9], present a variant of TSQL2 (drawn from the design of a language called Applied TSQL2, or ATSQL2) that addresses these concerns. In particular, the `VALID` clause was moved from inside the `SELECT` to before it. In this way, the comments of the ANSI committee resulted in a better language design, while retaining the bulk of the constructs of the original definition.

**7.** *What if multiple times are required in a table?*

[8] goes into some detail justifying why TSQL2 adopts precisely one valid time and one transaction time. However, the user is free to include datetime, interval, and period columns in a table that has temporal support. If the user desires all temporal information as columns, then the table with temporal support can be converted to a conventional table, with multiple datetime columns, either in a view or within a particular query, thereby giving each time-oriented column equal status. Alternatively, if the user desires explicit columns, then temporal support can be added to such tables within a particular query, to simplify that query.

**8.** *Won't these features be hard to implement?*

The answer is surprising. In terms of table representation, the implicit timestamps can be stored internally as an additional column. Hence, existing storage structures and access methods apply without change. Nonsequenced queries and modifications on tables with temporal support turn out to be very similar to existing conventional queries and modifications, making them easy to implement.

Sequenced queries can be implemented using a temporal algebra. Expressions in this temporal algebra can be converted to the conventional Codd algebra [9]. Hence, the underlying query optimizer and evaluator need not be changed. Some new code will be required in the query analysis portion of the DBMS to support the additional functionality available to the user.

A publicly-available prototype demonstrates feasibility [10]. This prototype may be down-loaded from `FTP.cs.arizona.edu` in the `timecenter` directory.

Recall also that these new facilities will be added to SQL/Temporal, and so their implementation will not be required for compliance with the SQL3 standard. Support of these facilities will be completely optional, though highly desired by users.

**9.** *What about the new features introduced in SQL3?*

The change proposals have been carefully written to minimize undesirable interactions with the underlying conventional query language. Existing queries on tables with temporal support access the current state of the table, enabling non-temporal applications to continue to work unchanged when temporal support is added. Nonsequenced queries (views, cursors, modifications, etc.) utilize the exact semantics of the underlying query language, making the valid time available as simply another column. This means that nonsequenced queries will work with whatever is added to SQL3.

Sequenced queries (views, etc.) apply the semantics of the underlying query language to each point in time. So the sequenced version of the query "List the employees who are not managers" applies the query to the employees on January 1 to get the result on January 1, then to the employees on

January 2 to get the result on January 2, etc. (This can be done much more efficiently than actually executing the query for each time point [9].) The semantics of sequenced queries is also in terms of the underlying query language. If a new construct, with new semantics, is added to SQL3, then that semantics will be applied on a point by point basis in a sequenced query.

In conclusion, the novel features of SQL3 should not in general present undesirable interactions with the features introduced in SQL/Temporal.

**10.** *Doesn't this break the relational model?*

The proposed changes extend the query and data manipulation language components of SQL, but do not impact the underlying data model, which remains one of tables with all information recorded as values in columns.

If a user wishes the timestamp of a table with valid-time support to be an explicit column, that is easy to specify, as a view or within the query. The same holds for tables with transaction-time support. If users only want current information from a table with temporal support, that is easy to specify (in fact, that is the default). If users wish to avail themselves of the above-listed advantages of tables with temporal support, that is also possible.

Again, it is not an either-or situation. Users are free to apply the facilities best suited to solve their problems. In fact, one can conceptualize tables with temporal support as being simply special "views" on conventional tables with explicit timestamp columns. Consider two tables: a table VT with an implicit timestamp and a snapshot table ST with identical columns as well as an explicit timestamp column. VT can be considered to be a special view of ST. Operations on VT will be defined (in General Rules) as a shorthand for equivalent, though more complex, operations on ST. Hence, tables with temporal support are just another way to query and modify conventional tables, and are thus fully consistent with the relational model.

In the next section, we discuss the staging of temporal constructs into SQL/Temporal. We indicate what will be proposed, and when.

# 3 SQL/Temporal Change Proposals

We identify three phases in the evolution of the SQL/Temporal part.

## 3.1 Initial Base Document

The goal of this phase was to establish a new part, called SQL/Temporal, and to include support for a new data type to augment `DATE`, `TIME`, `TIMESTAMP`, and `INTERVAL`. This data type, `PERIOD`, is now present in the base document.

## 3.2 Early Progression

The goal of this phase is to include the core concepts into SQL/Temporal. With this functionality, users will be able to express queries, modifications, views, and integrity constraints on time-varying data in a much more natural fashion than is possible with the current SQL3.

The intention is to have a modified version of the first change proposal (valid time), and an initial version of the other change proposal (transaction time), to the ANSI committee for its next meeting. These proposals build on each other, and it is important to understand each before going on. If acceptable proposals emerge from this meeting, then they could be discussed at the May ISO meeting.

For each change proposal, the major features to be proposed are listed.

- Change Proposal: Adding Valid Time
  - Tables with valid-time support
  - Temporal upward compatibility
  - Sequenced queries
  - Sequenced modifications
  - Sequenced views, cursors, integrity constraints
  - Nonsequenced queries
  - Nonsequenced modifications
  - Converting between tables with and without valid-time support
  - Accessing the valid time in the where clause and order clause

  Note that, as discussed above, using explicit timestamp columns violates temporal upward compatibility, and makes the specification of sequenced queries, modifications, views, cursors, and integrity constraints often very difficult.

  These powerful temporal capabilities are added in such a way that they can be used with straightforward clauses, predicates, and other appropriate constructs introduced into existing statements.

- Change Proposal: Adding Transaction Time
  - Tables with transaction-time support
  - Transaction sequenced queries and modifications
  - Transaction non-sequenced queries
  - Transaction time in views and integrity constraints
  - Accessing the transaction time

  Transaction sequenced queries are entirely analogous to, and orthogonal to, valid sequenced, in syntax and semantics. This change proposal will provide a straightforward application of valid-time concepts to this new concept of transaction time. The syntactic additions are quite minimal.

# 4 Later Progression

The goal of this progression is to round out SQL/Temporal with additional language features from TSQL2 that will be very helpful to large classes of users. These facilities should be considered only when the facilities of the early progression have stabilized.

The facilities in this portion are deferred for one or both of two reasons: the syntactic changes are more involved or the facilities are less central, though still important.

The design has been fully worked out in TSQL2 [8]; these additions are compatible with both the changes in the Early Progression and other features of SQL3.

The order of these change proposals has not been determined; they are listed alphabetically. Most of the change proposals are independent of each other, with the exception of temporal indeterminacy and user-defined time granularities, which are closely integrated.

For details, consult the indicated chapter of the TSQL2 book [8].

- Change Proposal: Event tables (cf., Chapter 16 of [8])
- Change Proposal: The From clause (Chapter 12)

  Temporal coalescing combines overlapping or adjacent periods into maximal periods. Restructuring provides very convenient syntactic sugar in the from clause. No new reserved words are added.

- Change Proposal: Now-relative values in the database (Chapter 20)

  Now-relative values add the ability to store `CURRENT_TIMESTAMP` as a data value in tables.

- Change Proposal: Schema versioning (Chapter 22)
- Change Proposal: Temporal grouping for aggregates (Chapter 21)
- Change Proposal: Temporal indeterminacy (Chapter 18)
- Change Proposal: User-defined time granularities (Chapter 19)

  This will include user-defined literal formats and multi-lingual support for time literals.

- Change Proposal: Vacuuming (Chapter 23)

Applications managing time-varying data impose requirements that are not met by SQL. Fortunately, there now exists a carefully designed solution, developed by a team of temporal database experts, who are fully knowledgeable about the extensive prior research. This solution has a formal semantics, a detailed discussion of the design decisions [8], and a prototype implementation [10]. It is a comprehensive, consistent, and well documented design.

The designers are willing to cooperate with the SQL3 committees to transition the insights of the long history of temporal database research into SQL3. This presents an unprecedented opportunity for the research and standards communities to work together to provide facilities that allow users to increase their productivity and simplify the expression of the database design, of queries, and of modifications. In this way, database technology is rendered even more applicable to and supportive of virtually all database applications.

# 5 Acknowledgments

The comments of John Bair, Mike Böhlen, Amelia Carlson, Curtis Dyreson, Len Gallagher, Christian S. Jensen and Jim Melton improved this document considerably. Extensive discussions with Hugh Darwen and Mike Sykes clarified many of the topics presented here.