On the Issue of Valid Time(s) in Temporal Databases

Stavros Kokkotos, Efstathios V. Ioannidis, Themis Panayiotopoulos, Constantine D. Spyropoulos
Institute of Informatics and Telecommunications, N.C.S.R. "Demokritos"
15310 Aghia Paraskevi, Greece
Tel: + 30-1-651 0310, Fax: + 30-1-653 2175
e-mails: {stavros,stathis,themisp,costass}@iit.nrcps.ariadne-t.gr

Abstract

Recent research activities in the area of Temporal Databases have revealed some problems related to the definition of time. In this paper we discuss the problem arising from the definition of valid time and the assumptions about valid time, which exist in current Temporal Database approaches. For this problem we propose a solution, while we identify some consistency problems that may appear in Temporal Databases, and which require further investigation.

1. Introduction

Since the dawn of the era of the Temporal Databases there exist three definitions regarding time in databases [3,6,7,8,9,10]: transaction time is the time when something is recorded in the database, valid time is the time that something happens in the modelled world, and user time is anything else concerning time. The common assumption is that for every tuple, or for every field there exist one transaction, one valid and maybe several types of user time. The transaction time reflects the history of the database, while the valid time reflects the history of the world.

Recently there have been some second thoughts, regarding the feasibility of having only one valid time for every tuple or field [2,7,8]. It seems that in some cases there exist many candidate valid times. The actual valid time is always a matter of semantics.

In chapter 2 we present some examples in order to clarify the valid time problem, while in chapter 3 we propose a solution that enables the database to manage more than one valid times. In chapter 4 we describe some as yet unsolved problems arising from the existence of more valid times, while in chapter 5 we present our concluding remarks.

2. The Problem with Valid Time

Consider the case of the record of an airline return ticket. This ticket has static information (i.e. *code*, *issuer*, *customer name*, *airline code*, *flight numbers*, *cities*, *price*, etc.) as well as several temporal information (i.e. *transaction date*, *issue date*, *scheduled date of first flight*, *scheduled date of return flight*, *actual date of first flight*, *actual date of return flight*, *ticket validity period for the first flight*, *ticket validity period for the return flight*). In this case the analysis of the various temporal information of the airline ticket is as follows:

- The *transaction date* is the date that the ticket was recorded into the airline's database. There seems to be no doubt that this is the transaction time.
- *Issue date* is the date that the travel agent issued the ticket.
- *Scheduled date of first flight* and *scheduled date of return flight* are the dates written upon the ticket, that the customer is scheduled to fly.
- *Actual date of first flight* and *actual date of return flight* are the dates that the customer took the plane on both parts of the trip. These may be different from the scheduled dates, because the airline agreed to put the customer to other flights.
- *Ticket validity period for the first flight* and *Ticket validity period for the return flight* are the time periods, during which the ticket is valid. When these periods end the customer looses his right to fly.

Each of these dates, except the *transaction date*, stamps a change that happens, or is scheduled to happen in the modelled world, the world of airlines, planes and customers. Regarding the valid time there exist several opinions. One interpretation is that the *issue date* is the valid time. But is this the most important date? The customer, or the airline administration, do not often ask about the *issue date* of a ticket. Instead they ask about the flight date. But we have four different flight dates: two scheduled (go and return) and two actual. Which one should be the valid time? Furthermore, we have two other time periods or intervals, ironically named validity periods, which compete for the title of valid time. The decision must be taken by the system developer. Obviously this is not a satisfactory solution, because we assign the label "valid time", followed by certain semantics and handled with a certain methodology, to a date or period, semantically similar to other dates or periods that we call "user times", which are handled in a different way.

An important distinction between field-bound and tuple-bound times must be made. As field-bound time we define a temporal attribute that characterises a field of a table. For example the *scheduled date of return flight*, of the airline ticket example, characterises the *return flight code* field. A tuple-bound time, such as the *issue date*, characterises the entity ticket, that is the whole ticket

tuple. Since the primary key characterises a whole tuple, we can transform the tuple-bound times into field-bound if we assume that they characterise the primary key of the tuple. So, in the airline ticket example, we assume that the *issue date* characterises the *ticket code* field.

The above distinction between field-bound and tuple-bound times leads to the interesting issue of field and tuple versioning data representation. A relational database supports field versioning if every field of a table can be associated to time attributes separately from the other fields. This leads to a representation of variable length records, because for each tuple there may be several field changes, which all belong to the same tuple. Tuple versioning is supported by a relational database when every time attribute characterises the set of fields in a tuple as a whole. For every change of any field value a new tuple is created. The tuple versioning approach is closer to the current relational approach, but in the case of temporal databases it results in data redundancy. The methodology followed by a temporal database, tuple or field versioning, is a matter of convention and design principles. In the following we will discuss both the tuple and field versioning methodologies.

3. The Proposed Solution

It seems that we must accept the situation of having more than one valid times for a tuple or field. According to our opinion there exist 3 solutions:

- For each different valid time to create a new table, with the primary key and the relative with the valid time fields from the master table.
- To have only user times, without valid one.
- To only valid times, with labels that define their semantics, without user times.

Using the first solution in the airline ticket example, and for tuple versioning, we have to create the following tables. Because of the tuple versioning representation, each table has only one valid time.

— The master table, which has all the flight data except the *scheduled flight code*. Primary key is the *ticket code*. Valid time is the *issue date*.
— The scheduled first flight table, with primary key the *ticket code*, and only one other field, the *scheduled first flight code*. Valid time is the *scheduled date of first flight*.
— The scheduled return flight table, with primary key the *ticket code*, and only one other field, the *scheduled return flight code*. Valid time is the *scheduled date of return flight*.
— The actual first flight table, with primary key the *ticket code*, and only one other field, the *actual first flight code*. Valid time is the *actual date of first flight*.
— The actual return flight table, with primary key the ticket code, and only one other field, the actual return flight code. Valid time is the actual date of return flight.
— The first flight validity period table, with its sole field the primary key ticket code. Valid time is the ticket validity period for the first flight.
— The return flight validity period table, with its sole field the primary key ticket code. Valid time is the ticket validity period for the return flight.

Using the first solution in the airline ticket example, and for field versioning, we have the following tables. Here, because of the field versioning, we may have more than one valid times in every table, but only one valid time for every field.

— The master table, which has all the flight data. Primary key is the ticket code. The valid time of the ticket code field is the issue date. The valid time of the first flight code field is the scheduled date of first flight. The valid time of the return flight code field is the scheduled date of return flight. Valid time of the actual first flight code field is the actual date of first flight. The valid time of the actual return flight code field is the actual date of return flight.
— The ticket validity period table, with primary key the ticket code, and two other fields, the first flight code and the return flight code. Valid time of the first flight code field is the ticket validity period for the first flight. The valid time of the return flight code field is the ticket validity period for the return flight.

This solution can not be accepted, for either tuple or field versioning, because it increases the number of the tables, resulting in a more complicated database administration. Furthermore the queries become slower and less effective, due to the need for multiple joins. There is no reason for distributing the information to more than one tables, except from the need to keep only one valid time per tuple or field. Each of the above tables has the same primary key, so they can be merged into one table, without loss of information.

The second and third approaches result in the database having one transaction time and a set of either user or valid times, each with a label that defines their semantics. These two approaches, if we disregard the name of the times (user or valid), are actually the same: We have a set of labelled times.

Since the first approach leads to unacceptable information distribution we propose that a temporal database has one transaction time, automatically maintained by the system, and a set of labelled valid times, provided by the user.

The airline ticket example, for the tuple versioning approach, has one transaction time and seven valid times labelled *issue date*, *scheduled date of first flight*,

scheduled date of return flight, *actual date of first flight*, *actual date of return flight*, *ticket validity period for the first flight*, and *ticket validity period for the return flight*. We could consider that every different valid time belongs to a different time line, labelled by the valid time label. So all the *issue date* times belong to the "issue date" time line.

For the field versioning approach the *issue date* characterises the *ticket code* field, the *scheduled date of first flight* and *ticket validity period for the first flight* characterise the *first flight code field*, the *actual date of first flight* characterises the *actual first flight code field*, the *scheduled date of return flight* and *ticket validity period for the return flight* characterise the *return flight code field*, while the *actual date of return flight* characterises the *actual return flight code field*.

To facilitate the proposed valid time solution, the temporal SQL [1,5,11] should be extended in order to provide facilities to identify the appropriate valid times in a query, by their labels. We propose that the "as-of" operator of a temporal SQL (transaction time) remains as it is, while the "when" operator (valid time) is extended with a "timelabel" operator, that defines the time line on which the "when" operates.

In a tuple versioning database, in order to ask for the *ticket codes* and *actual return date* of return flight OA452 flying before date 1/1/95, that actually flied on a different date than the scheduled, as the database knew these to be on 2/2/95, the user has to pose the following query:

```
SELECT Code, TIMELABEL(ActualDateRetFlight)
FROM Ticket
  WHERE   SchedRetFlightCode = "OA452"
  WHEN    TIMELABEL(SchedDateRetFlight) <
                #1/1/95#
  AND     TIMELABEL(ActualDateRetFlight) ≠
                NULL
  AND     TIMELABEL(SchedDateRetFlight) ≠
                TIMELABEL(ActualDateRetFlight)
  AS-OF   #2/2/1995#;
```

The NULL value in a time attribute means that the user has not provided a value for this attribute. In our case we have to be sure that this ticket actually was used (time ≠ NULL).

In a field versioning database the above queries will be expressed as:

```
SELECT Code,
ActualRetFlightCode.TIMELABEL(ActualDateRetFlight)
FROM Ticket
  WHERE   SchedRetFlightCode = "OA452"
  WHEN
  SchedRetFlightCode.TIMELABEL(SchedDateRetFlight)
                < #1/1/95#
  AND
  ActualRetFlightCode.TIMELABEL(ActualDateRetFlight)
                ≠ NULL
  AND
  SchedRetFlightCode.TIMELABEL(SchedDateRetFlight)

  ActualRetFlightCode.TIMELABEL(ActualDateRetFlight)
  AS-OF   #2/2/1995#;
```

The TIMELABEL operator, in a field versioning database, always refers to a field, because there may be valid times with the same label, each assigned to a different field.

4. Discussion

The proposed solution is not a panacea. There are still issues that need further discussion. For example, the introduction of time in a database may lead to inconsistency problems [2,7,8].

Consider the following situation in a banking environment: Customer C has a joint money account in a bank together with his wife W. At 8:00 of the 1/1/94 (valid time) the account holds $15000. At 12:00 of the same day (valid time) the customer C deposits $5000 in the account. The history of the account balance is as displayed in figure 1.

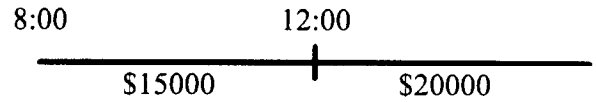

Figure 1: History of the balance of a bank account

At 10:00 of the same day 1/1/94 (valid time) the wife of C, that is customer W, made a withdrawal of $5000. This withdrawal was not recorded immediately, due to problems of the bank's network. The transaction was recorded at 14:00 after A's deposit of 12:00. The database must now be modified by inserting the balance $10000 valid from 10:00 until 12:00, as displayed in figure 2.

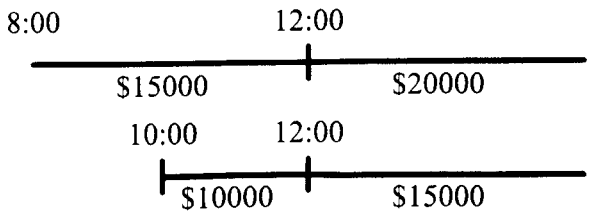

Figure 2: Modification of the balance history of a bank account

If the database allows this insertion, then for the period 10:00-12:00 there exist 2 different balance amounts in the bank account, $15000 and $10000. It is obvious to us that the correct answer is $10000 because the $10000 balance is stamped by a later transaction time (10:00). The same discussion can be made for the balances of $20000 and $15000 for the period later than 12:00. The problem that arises is a matter of the following decision. Should we allow the database to draw this conclusion by itself, or should we consider this insertion as an inconsistency, and ask the application or the user to explicitly modify the validity period of the $15000 balance to 8:00-10:00, before inserting the $10000 balance update? In other words should we allow the database to alter our data without consulting us or our application? This is an issue that needs further investigation, because there are no standard rules, other than common sense that is absent from a database system. This inconsistency problem may be resolved by extending the table definition semantics, in order to include guidelines for the database, concerning the handling and freedom allowed for each of the multiple valid times.

Another issue that has been described [4,7,8], but is not discussed here, is the issue recording subsequent changes to table definitions connecting table definitions with time. This case presents many interesting problems, such as the situation where a field changes type, and we must keep the old tuples with the old field type, while the new ones will be of the new type. Another problem arises in the case where a query spans two table definition periods, with different fields, field types and field names.

5. Conclusions

In this short paper we defined the existence of a serious problem, when the approach of a single valid time per tuple or field is adopted. This problem is an important one and leads to different approaches in the database schema, according to the semantics that the developer introduces.

We presented three different solutions, from which one is unacceptable, due to increased complexity to the database schema and the transaction handling, while the other two, being equivalent, can be merged into the final proposed solution.

Our proposed solution is to abolish the notion of user time, and allow for multiple valid times, each one belonging to a labelled time line. Thus the semantics of these times does not suffer, while the user may select the queries, to contain any needed valid time. For this reason an extension to the "when" operator (valid time) of the temporal SQL is proposed, that identifies the time line to which the "when" will be performed.

Very interesting consistency problems arise from the recording of time in databases, both at the data management and the data definition level. These issues need to be further investigated.

References

[1] I. Ahn, SQL+T: A Temporal Query Language. in *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[2] E.V. Ioannidis, C.D. Spyropoulos et al, TIMERx: a Kernel for Managing Temporal Reference in a RDBMS. Working Paper, N.C.S.R. "Demokritos" Aghia Paraskevi, Greece, 1994.

[3] C.S. Jensen et al, A Consensus Glossary of Temporal Database Concepts. Technical Report R 93-2035, Aalborg University, Nov. 1993.

[4] S. Kokkotos and C.D. Spyropoulos, A Framework for Developing Temporal Databases. in *Proceedings of the DEXA '94 International Conference*, Athens, Greece, Sep. 1994, *Lecture Notes in Computer Science 856*, Springer-Verlag, Berlin, 1994, 236-245.

[5] N.A. Lorentzos and Y.G. Mitsopoulos, IXSQL: An Interval Extension to SQL. in *Proceedings of the International Workshop on an Infrastructure for Temporal Databases*, Arlington, TX, June 1993.

[6] E. McKenzie and R. Snodgrass, An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys* 23(4), Dec. 1991, 501-543.

[7] N. Pissinou, R. Snodgrass et al, Towards an Infrastructure for Temporal Databases, Report of an Invitational ARPA/NSF Workshop. Technical Report 94-01, University of Arizona, Tucson, AZ, Mar. 1994.

[8] N. Pissinou, R. Snodgrass et al, Towards an Infrastructure for Temporal Databases, Report of an Invitational ARPA/NSF Workshop. *SIGMOD RECORD*, 23(1), Mar. 1994, 35-51.

[9] R. Snodgrass and I. Ahn, A Taxonomy of Time in Databases. in *Proceedings of the ACM International Conference on Management of Data*, Austin, TX, May 1985, 236-246.

[10] R. Snodgrass and I. Ahn, Temporal Databases. *IEEE Computer* 19(9), Sep. 1986, 35-42.

[11] R. Snodgrass, An Overview of TQUEL. *Chapter 6, Temporal Databases: Theory, Design and Implementation,* Benjamin/Cummings, 1993, 141-182.